Sun WorkShop TeamWare User's Guide

Forte Developer 6 update 2
(Sun WorkShop 6 update 2)

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

Part No. 806-7992-10
July 2001, Revision A

Send comments about this document to: docfeedback@sun.com

Please
Recycle

Adobe PostScript

Important Note on New Product Names

As part of Sun's new developer product strategy, we have changed the names of our development tools from Sun WorkShop™ to Forte™ Developer products. The products, as you can see, are the same high-quality products you have come to expect from Sun; the only thing that has changed is the name.

We believe that the Forte™ name blends the traditional quality and focus of Sun's core programming tools with the multi-platform, business application deployment focus of the Forte tools, such as Forte Fusion™ and Forte™ for Java™. The new Forte organization delivers a complete array of tools for end-to-end application development and deployment.

For users of the Sun WorkShop tools, the following is a simple mapping of the old product names in WorkShop 5.0 to the new names in Forte Developer 6.

Old Product Name	New Product Name
Sun Visual WorkShop™ C++	Forte™ C++ Enterprise Edition 6
Sun Visual WorkShop™ C++ Personal Edition	Forte™ C++ Personal Edition 6
Sun Performance WorkShop™ Fortran	Forte™ for High Performance Computing 6
Sun Performance WorkShop™ Fortran Personal Edition	Forte™ Fortran Desktop Edition 6
Sun WorkShop Professional™ C	Forte™ C 6
Sun WorkShop™ University Edition	Forte™ Developer University Edition 6

In addition to the name changes, there have been major changes to two of the products.

- Forte for High Performance Computing contains all the tools formerly found in Sun Performance WorkShop Fortran and now includes the C++ compiler, so High Performance Computing users need to purchase only one product for all their development needs.
- Forte Fortran Desktop Edition is identical to the former Sun Performance WorkShop Personal Edition, except that the Fortran compilers in that product no longer support the creation of automatically parallelized or explicit, directive-based parallel code. This capability is still supported in the Fortran compilers in Forte for High Performance Computing.

We appreciate your continued use of our development products and hope that we can continue to fulfill your needs into the future.

Contents

Figures

Tables

Before You Begin

The *Sun WorkShop TeamWare User's Guide* describes how to use the Sun Workshop™ TeamWare code management tools. This manual is intended for the software developer, but can be used by anyone involved in team development of a product, including integrators, administrators, and release engineers.

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

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

This manual assumes some understanding of the Solaris™ Operating Environment and UNIX® commands. You need not have previous experience with the Source Code Control System (SCCS).

Chapter 9 and Chapter 10 are a supplement to the Solaris Operating Environment `make` documentation. They describe how to use Building and Distributed Make to improve the process of building programs and make it more efficient. Use these chapters if you maintain programs using the `make` utility and wish to speed up the build process. These chapters assume that you are familiar with the standard `make` utility and that you are familiar with programming constructs and processes.

How This Book Is Organized

Chapter 1 provides a full introduction to the Sun WorkShop TeamWare product.

Chapter 2 presents step-by-step instructions on how to create and manage workspace.

Chapter 3 provides step-by-step instructions on how to customize Configuring and perform more advanced tasks on workspaces.

Chapter 4 provides step-by-step instructions on how to grant or deny permission to perform workspace transactions.

Chapter 5 provides step-by-step instructions on how to check out and put back files.

Chapter 6 provides step-by-step instructions on how to resolve the differences between files in the Merging tool.

Chapter 7 provides step-by-step instructions on how to view file history, move or rename files, and customize Versioning.

Chapter 8 provides step-by-step instructions on how to create and use freezepoints.

Chapter 9 provides step-by-step instructions on how to build specific targets, as well as hints on fixing build errors.

Chapter 10 describes `dmake`, a tool that allows you to distribute builds over several hosts concurrently. The operation of `dmake` is described, and instructions given on how to distribute your build efficiently.

Chapter 11 provides information on how to use Sun WorkShop TeamWare commands at the command line, and also lists mouse and keyboard shortcuts.

Chapter 12 lists metadata files, describes how Configuring manipulates SCCS history files during file transfer transactions, and explains SCCS Mergeable IDs (SMIDs) and SCCS delta IDS (SIDs).

Appendix A lists error messages and warnings. Each message is defined, and a possible solution is provided.

Appendix B lists some common problems with Sun WorkShop TeamWare and their solutions.

The Glossary provides an explanation of the special terms used in this manual.

Typographic Conventions

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

Shell Prompts

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

Supported Platforms

This Sun WorkShop™ release supports versions 2.6, 7, and 8 of the Solaris™ *SPARC™ Platform Edition* and Solaris™ *Intel Platform Edition* operating environments.

Accessing Sun WorkShop Development Tools and Man Pages

The Sun WorkShop product components and man pages are not installed into the standard `/usr/bin/` and `/usr/share/man` directories. To access the Sun WorkShop compilers and tools, you must have the Sun WorkShop component directory in your `PATH` environment variable. To access the Sun WorkShop man pages, you must have the Sun WorkShop man page directory in your `MANPATH` environment variable.

For more information about the `PATH` variable, see the `csh`(1), `sh`(1), and `ksh`(1) man pages. For more information about the `MANPATH` variable, see the `man`(1) man page. For more information about setting your `PATH` and `MANPATH` variables to access this release, see the *Sun WorkShop 6 update 2 Installation Guide* or your system administrator.

Note – The information in this section assumes that your Sun WorkShop 6 update 2 products are installed in the `/opt` directory. If your product software is not installed in the `/opt` directory, ask your system administrator for the equivalent path on your system.

Accessing Sun WorkShop Compilers and Tools

Use the steps below to determine whether you need to change your `PATH` variable to access the Sun WorkShop compilers and tools.

To Determine If You Need to Set Your `PATH` Environment Variable

1. **Display the current value of the `PATH` variable by typing:**

```
% echo $PATH
```

2. **Review the output for a string of paths containing `/opt/SUNWspro/bin/`.**

   If you find the path, your `PATH` variable is already set to access Sun WorkShop development tools. If you do not find the path, set your `PATH` environment variable by following the instructions in the next section.

To Set Your PATH Environment Variable to Enable Access to Sun WorkShop Compilers and Tools

1. **If you are using the C shell, edit your home `.cshrc` file. If you are using the Bourne shell or Korn shell, edit your home `.profile` file.**
2. **Add the following to your `PATH` environment variable.**

   `/opt/SUNWspro/bin`

Accessing Sun WorkShop Man Pages

Use the following steps to determine whether you need to change your `MANPATH` variable to access the Sun WorkShop `man` pages.

To Determine If You Need to Set Your MANPATH Environment Variable

1. **Request the `workshop` man page by typing:**

```
% man workshop
```

2. **Review the output, if any.**

   If the `workshop`(1) man page cannot be found or if the man page displayed is not for the current version of the software installed, follow the instructions in the next section for setting your `MANPATH` environment variable.

To Set Your MANPATH Environment Variable to Enable Access to Sun WorkShop Man Pages

1. **If you are using the C shell, edit your home `.cshrc` file. If you are using the Bourne shell or Korn shell, edit your home `.profile` file.**
2. **Add the following to your `MANPATH` environment variable.**

   `/opt/SUNWspro/man`

Accessing Sun WorkShop Documentation

You can access Sun WorkShop product documentation at the following locations:

- **The product documentation is available from the documentation index installed with the product on your local system or network.**

  Point your Netscape™ Communicator 4.0 or compatible Netscape version browser to the following file:

  `/opt/SUNWspro/docs/index.html`

  If your product software is not installed in the `/opt` directory, ask your system administrator for the equivalent path on your system.

- **Manuals are available from the docs.sun.com℠ Web site.**

  The `docs.sun.com` Web site (`http://docs.sun.com`) enables you to read, print, and buy Sun Microsystems manuals through the Internet. If you cannot find a manual, see the documentation index installed with the product on your local system or network.

Accessing Related Documentation

You can access documentation related to the subject matter of this book in the following ways:

- **Through the Forte TeamWare Release Notes**.

  The Forte TeamWare release notes are available in:

  `/opt/SUNWspro/READMEs/teamware`

- **Through the Forte TeamWare Online Help.**

  Choose Help ➤ About Documentation in any Forte TeamWare tool window.

- **Through the Forte TeamWare Quick Tour.**

  The interactive Forte TeamWare Quick Tour provides a high-level overview of Forte™ TeamWare's basic model and features. It is accessible from the Help menu (choose Help ➤ TeamWare Quick Tour) in any Sun WorkShop TeamWare window.

The following table describes related documentation that is available through the `docs.sun.com` Web site.

Document Collection	Document Title	Description
Numerical Computation Guide Collection	*Numerical Computation Guide*	Describes issues regarding the numerical accuracy of floating-point computations.
Solaris 8 Reference Manual Collection	See the titles of man page sections.	Provides information about the Solaris operating environment.
Solaris 8 Software Developer Collection	*Linker and Libraries Guide*	Describes the operations of the Solaris link-editor and runtime linker.
Solaris 8 Software Developer Collection	*Multithreaded Programming Guide*	Covers the POSIX and Solaris threads APIs, programming with synchronization objects, compiling multithreaded programs, and finding tools for multithreaded programs.

Ordering Sun Documentation

You can order product documentation directly from Sun through the `docs.sun.com` Web site or from Fatbrain.com, an Internet bookstore. You can find the Sun Documentation Center on Fatbrain.com at the following URL:

`http://www.fatbrain.com/documentation/sun`

Sending Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. Email your comments to Sun at this address:

`docfeedback@sun.com`

CHAPTER 1

Introduction to Sun WorkShop TeamWare

This chapter provides an overview of Sun WorkShop™ TeamWare. The following topics are discussed:

- What Sun WorkShop TeamWare Does
- Basic Concepts
  - Why Use Sun WorkShop TeamWare?
  - Parent and Child Workspaces
  - Source Code Control System (SCCS)
- Sun WorkShop TeamWare Models
  - The Copy-Modify-Merge Model
  - The Team-Project Model
- Working With Sun WorkShop TeamWare
- Sun WorkShop TeamWare Scenarios
  - Joining an Existing Team
  - Setting Up a Sun WorkShop TeamWare Environment

In the following chapters, you'll find detailed instructions for using Sun WorkShop TeamWare tools and features.

What Sun WorkShop TeamWare Does

Sun WorkShop TeamWare is a source management product designed for use by a team of people who develop software concurrently. Sun WorkShop TeamWare tools save time and increases your organization's productivity by simplifying source code management. More specifically, Sun WorkShop TeamWare lets you:

- Configure your working directories and subdirectories (or folders) into workspaces to suit the phases of your project and the structure of your team.

- Work on individual versions of the same file, but also ensure that everyone has the latest version in their own individual workspaces.
- Inside a single workspace, lock a file for editing to prevent unmanaged conflicts.
- Inspect and selectively merge versions of a single file that was edited in different workspaces.
- Find and "freeze" a particular version of a workspace, archiving it for later use.
- Build an application for release from selected files and directories—either locally (serially or in parallel) or across distributed systems.
- Automatically notify team members of each action that has been taken on a file.

Sun WorkShop TeamWare's version-control capability is based on an underlying program called SCCS (Source Code Control System). The workspaces you configure with Sun WorkShop TeamWare tools act only upon files that have been placed under SCCS control. If you have files under the RCS source-code control system, they can be migrated to Sun WorkShop TeamWare.

Basic Concepts

If you are a Sun WorkShop TeamWare user, you should understand the development model it is based on, *Copy-Modify-Merge*. You should also understand the relationships between workspaces, based on the concept of *parent* and *child* workspaces. Sun WorkShop TeamWare works only on files under UNIX Source Code Control System (SCCS) version control; you should be familiar with SCCS. Each of these is briefly described in the following sections.

Why Use Sun WorkShop TeamWare?

The hardest part of most large software development projects is coordinating the work of developers who share common and interdependent files.

If developers have private copies of the source code, the changes they make to the source base are difficult to track when all of the code is finally merged. One solution is to allow serial access to the common files, one developer at a time. Unfortunately, when only one programmer at a time has access to the code, a bottleneck occurs.

Sun WorkShop TeamWare supports coordinated parallel development, because it lets you create one or more isolated, private workspaces for each developer. Each developer copies project files from a central workspace into his or her own private workspace, makes changes to files, and then copies those changes back to the central workspace.

Parent and Child Workspaces

Your team does its work in directories (or folders) and files. To gain the advantages of Sun WorkShop TeamWare, place all your working directories in one high-level directory. You then use Sun WorkShop TeamWare to transform that directory hierarchy into a workspace. Sun WorkShop TeamWare uses these tools:

- *Configuring* – Forms intelligent connections between workspaces that are owned by different Sun WorkShop TeamWare users. *Configuring* also maintains a history of the workspaces and all of the transactions performed.
- *Versioning* – Maintains a history of the files and the deltas to each file.
- *Merging* – Protects against changes to files overwriting each other.
- *Freezepointing* – Captures a "snapshot" of the workspace.
- *Building* – Combines files into a working application.

Sun WorkShop TeamWare converts your high-level directory into an intelligent workspace. When you create a new workspace from a copy of a workspace, a special relationship is created between the first workspace and the new copy. The first workspace is considered the *parent* of the newly created *child* workspace.

Rather than risk corrupting the master files in the parent workspace, each team member works on copies of those files in his or her own child workspace. Sun WorkShop TeamWare lets team members easily copy directories back and forth between their child workspaces and the parent workspace.

A parent workspace can have many of child workspaces, one or more for each team member. Team members populate their child workspace(s) with the directories and files that they need. A particular child workspace can contain a copy of every directory and every file in the parent workspace, or only a subset of the parent's contents.

In a complex project encompassing many levels, one workspace may be the parent of some workspaces and the child of another workspace.

Source Code Control System (SCCS)

Sun WorkShop TeamWare recognizes only files under the Source Code Control System (SCCS). Each time you check out a file, change it, and check it back in, SCCS keeps track of the changes. The differences between two versions of a file is known as a *delta*. Sun WorkShop TeamWare manages files based on SCCS deltas. When you edit, move or copy a file, Configuring copies or merges the file's SCCS history file. The way Sun WorkShop TeamWare manipulates and merges SCCS history files is described in "How Configuring Merges Files" on page 161.

See the *Solaris Programming Utilities Guide* for a description of SCCS.

Sun WorkShop TeamWare Models

The following two models show how Sun WorkShop TeamWare used in organizations:

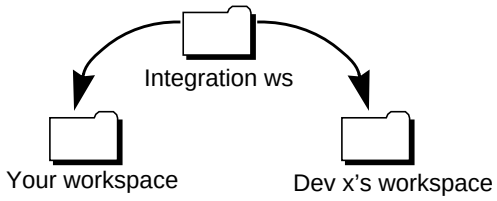
- The Copy-Modify-Merge model, which is the basic idea behind Sun WorkShop TeamWare
- Sun WorkShop TeamWare's user model for teams and projects—at set-up time and day-to-day

The Copy-Modify-Merge Model

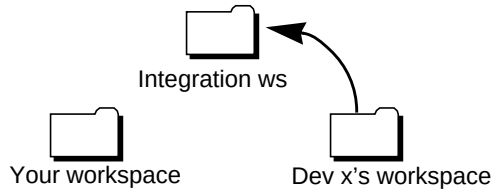
Sun WorkShop TeamWare configuration management tools are based on a concurrent-development model called Copy-Modify-Merge. When you use these tools in your daily work on project files, you iterate this basic pattern:

1. *Bring over* (copy) the latest version of a file or directory from the parent workspace.
2. *Modify* a file from that directory inside your own child workspace.
3. *Put back* that file to the parent workspace. If someone else has worked on another copy of the same file and put it back from his or her own workspace to the parent, you can selectively *merge* the two sets of changes.

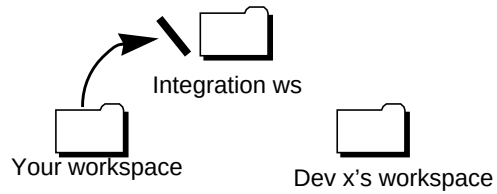
In the following Copy-Modify-Merge example, you see a common software development scenario in which two people are working simultaneously on the same or related parts of a project.



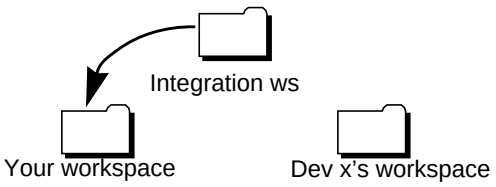
Both you and Developer x copy the same file from the project integration workspace to your separate, individual workspaces.



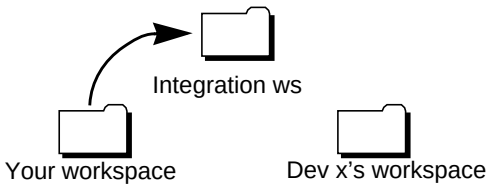
Developer x changes the file and copies the changed file back into the integration workspace.



You modify the same file in your workspace and attempt to copy the file back into the integration workspace. TeamWare Configuring blocks your attempt to copy, because it would overwrite Developer x's changes.



Configuring informs you of the conflicting changes. You copy the file containing Developer x's changes from the integration workspace to your workspace.



With Configuring's assistance, you resolve the conflicts, merge the changes, test the changes, and successfully copy the file back to the integration workspace.

The Team-Project Model

The Copy-Modify-Merge model for an individual user fits into Sun WorkShop TeamWare's larger team-development model for a team. Over the life of a typical large-scale project, your team may need to set up and use a complex project structure and process to reflect your project's shape and schedule. Following is an example of such a structure and process—first, during set-up time, and second, as the day-to-day work continues.

Sun WorkShop TeamWare Environment Setup

In the beginning of a project, before team development starts, the following happens.

1. One team member—perhaps a team leader or a system administrator—sets up a group of parent workspaces to house separate phases and divisions of your project, such as:
   - *Alpha* – The workspace where your team's latest files reside
   - *Integration* – The workspace where team leaders combine everyone's work
   - *Team Workspaces* – Each engineering team has their own workspace
   - *Beta/Final* - Different versions of builds based on criteria such as project phase, release, platform, or locale
2. Before starting to work on files, each team member sets up his or her own child workspace in relation to the team workspace (*Dev1*, *Dev2*, *Dev3*).
3. Each team member brings over from the parent workspace a copy of the directories and files he or she will need to modify.

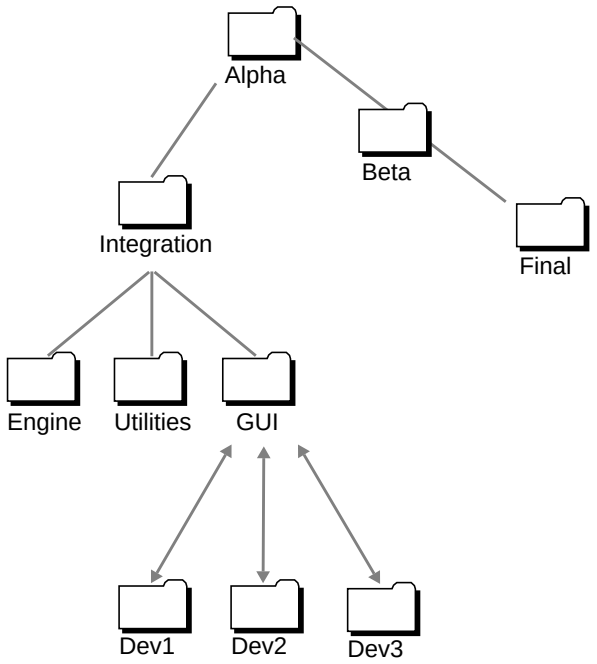


FIGURE 1-1 Setting up a Typical TeamWare Environment

Now the team's environment is set up for concurrent development. To continue the example, this team uses their workspace structure as described in the next section.

Day-to-Day Work in a Sun WorkShop TeamWare Environment

As the members of your development team work on files and directories, and as the team moves through the process of developing an application, they do the following tasks on a regular, iterative basis.

1. Team members bring over directories from the team parent workspace to their own child workspaces.
   - Each team member must do bringovers frequently. This practice keeps the team's changes freshly propagated throughout their workspaces, a necessity if everyone is to stay up-to-date.
   - Team members make changes in their individual child workspaces, and then put back the changed directories to the team parent workspace.
   - TeamWare automatically notifies the whole team (or whoever is on the notification list) when changes have been put back.

2. Team leaders check the changes that team members have put back. Team leaders then put back those changed directories from the team workspaces to the Integration workspace.
3. When notified that the files in the Integration workspace are ready, the buildmaster integrates all the files, builds the application, and puts back the resulting directories to the Alpha workspace.
4. When notified that Alpha workspace is ready, the software testers go to that workspace and test the application.
5. When the alpha phase of the product is complete, the Beta workspace is created as a child of the Alpha workspace. Then the Integration workspace is reparented to the Beta workspace.
6. Periodically, a team member sets aside or archives a particular "snapshot" of a workspace (a freezepoint) for later use. You can then extract a workspace from the freezepoint you created, with all of its contents reflecting that point in time.

Working With Sun WorkShop TeamWare

The following table briefly explains which Sun WorkShop TeamWare tool or feature you use for which kind of task.

TABLE 1-1 Uses for Sun WorkShop TeamWare Tools

Task	TeamWare Tool
Turn a directory into a workspace	Configuring
Create a child workspace from the parent workspace	Configuring
Bring over directories from the parent workspace to your child workspace	Configuring
Check out a file from your own workspace	Versioning
Modify and save the file	Versioning
Check the file back in to your workspace	Versioning
Create new files in a workspace	Versioning
Put back copies of the new and modified files to the parent workspace	Configuring
Merge conflicting versions of a file that two or three team members have modified in their own workspaces	Merging

TABLE 1-1 Uses for Sun WorkShop TeamWare Tools *(Continued)*

Task	TeamWare Tool
Put the resulting merged file back to the parent workspace	Configuring
Preserve a particular workspace version for later use	Freezepointing
Put together (make) files into a completed application for testing or release.	Building

Sun WorkShop TeamWare Scenarios

People have different interactions with of Sun WorkShop TeamWare depending on their responsibilities within their projects and teams. Consider the following two scenarios:

Joining an Existing Team

If you are the newest member of a team that has been using Sun WorkShop TeamWare configuration management tools for concurrent software development, and if their workspaces are already set up, you'll need to do the following.

To join an existing team:

1. **Find out the configuration of your team's parent workspace(s).**
2. **Find out your team's policies, processes, and schedules for builds.**
3. **Set up a child workspace for yourself, in which you will do your work.**
4. **Set up email notifications within Sun WorkShop TeamWare.**
5. **Bring over the directories you want from the parent workspace.**
6. **Check out each file to modify it; save it; and check it in again.**
7. **You can also add files to your child workspace and delete files from it. Place any new files under SCCS control.**
8. **When you are finished changing files, put back the whole directory to the parent workspace.**
9. **Your team is automatically notified of the changes you've made.**
10. **If you put back a file that another team member has already changed and put back, you must merge those changes and put the resulting file back to the parent.**

Setting Up a Sun WorkShop TeamWare Environment

If no Sun WorkShop TeamWare workspaces have been set up, and you are responsible for doing it, you'll need to do the following.

To set up an Sun WorkShop TeamWare environment:

1. **Decide how to configure your team's workspaces, based on criteria like the following:**
   - The project's discrete phases
   - The geographical distribution of your team, or the networks and file systems they use
   - The platforms on which and for which the application is being developed
   - The release structure of your project
2. **Create the parent workspace(s) for your project.**
3. **Notify your team of the workspaces you have created so that they can now create their child workspaces.**
4. **Set up and publish agreements within the team for:**
   - Regular bringovers
   - Putbacks
   - Periodic cutoffs (such as before weekly builds)
   - Periodic putbacks from lower parent workspaces to higher ones
   - Periodic builds
   - Freezepoints of builds or other milestone-related versions
5. **If you are also developing the application along with your team, set up your own child workspace(s) and use Sun WorkShop TeamWare tools on a regular basis as shown in "Joining an Existing Team" on page 17.**

CHAPTER 2

Managing Workspaces

To use Sun WorkShop TeamWare's features, you must put your files and directories into TeamWare workspaces. A *workspace* is a specially designated directory, its subdirectories, and the files contained in those directories. Using Sun WorkShop TeamWare tools, you manage the files in the workspace and the relationships with other workspaces. Use the Configuring tool to view workspaces, their relationships, and to execute commands on workspaces. This chapter shows you how to perform these basic tasks in Configuring:

- Starting Configuring
- Creating a Parent Workspace
- Creating Child Workspaces
- Propagating Changes Across Workspaces
- Undoing Changes to a Workspace
- Renaming or Moving Workspaces
- Deleting or Reconverting Workspaces
- Viewing Workspace History

Starting Configuring

To start Configuring, type the following at a command line:

```
% twconfig &
```

If you are running Sun WorkShop™, you can start TeamWare Configuring by:

- Clicking the TeamWare button on the tool bar in the Sun WorkShop main window
- Selecting TeamWare from the Tools menu

Note – Because Sun WorkShop product components and man pages do not install into the standard `/usr/bin/` and `/usr/share/man` directories, you must change your `PATH` and `MANPATH` environment variables to enable access to Sun WorkShop TeamWare tools. See "Accessing Sun WorkShop Documentation" on page 6.

When you start Sun WorkShop TeamWare Configuring, the Configuring window (see FIGURE 2-1) opens.

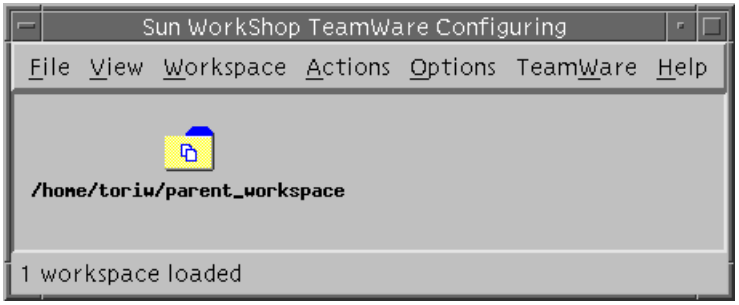


FIGURE 2-1 Configuring Window With Parent Workspace Loaded

TABLE 2-1 describes the Configuring window menus:

TABLE 2-1 Configuring Window Menus

Menu	Description
File	Provides commands for loading, unloading, and creating workspaces.
View	Provides commands to change how the workspaces appear in the workspace pane.
Workspace	Provides commands for managing workspaces.
Actions	Opens the Transactions dialog box, which provides commands for synchronizing files (bringovers, updates, and putbacks).
Options	Provides commands for setting the workspace and Configuring options.
TeamWare	Provides commands for starting other TeamWare tools, such as Versioning, Merging, and Freezepointing.
Help	Provides commands to start help, get a documentation road map, send comments to the Sun WorkShop TeamWare development team, and get the version number.

See the Sun WorkShop TeamWare online help for a complete list of these menus and their functions. For information on how to customize Configuring, see "Customizing Configuring Using Tool Properties" on page 49.

Creating a Parent Workspace

When you begin to use the Sun WorkShop TeamWare code management tools, you can start a new project or import existing project files into Sun WorkShop TeamWare. To do this, you must set up workspaces.

Start a project by creating a top-level (parent) workspace. You create a child workspace from this parent workspace by doing a special copy of the files called a *bringover*. You make changes in your own child workspaces, and later integrate your files with changes made by other developers. When you integrate changes, it is called a *putback*.

When you create a workspace, Sun WorkShop TeamWare creates a subdirectory, named `Codemgr_wsdata`, that stores information about the files in the workspace. There are two ways to create a parent workspace:

- Create an empty workspace and then populating it with a hierarchy of new directories and files
- Create a workspace from an existing hierarchy of files

Note – If you have existing project files, you should know the location (path name) of those files before you begin using Configuring.

Creating an Empty Workspace

To create a new empty workspace:

1. **Start Configuring.**
2. **Choose File ➤ Create Workspace.**
3. **In the Workspace Directory text box, type a workspace name.**

   If the directory does not already exist, Sun WorkShop TeamWare will create it.

4. **Click OK.**

   TeamWare Configuring creates a workspace in the directory you have specified and creates an icon for it in the Configuring window.

   When you create files, you will have to check the files in to the workspace. See "Adding Files to a Workspace" on page 67.

Creating a Workspace From Existing Files

If you have a directory containing the files that you want to make into a workspace, do the following to create a new workspace:

1. **Start Configuring.**
2. **Choose File ➤ Create Workspace.**
3. **In the Workspace Directory text box, type the path name to the directory that contains the files.**
4. **Click OK.**

   TeamWare Configuring creates a workspace in the directory you have specified and creates an icon for it in the Configuring window.

5. **Check in the files using Sun WorkShop TeamWare Versioning.**

   Sun WorkShop TeamWare recognizes only files that are under SCCS version control. If your files are not already under SCCS version control, see "Adding Files to a Workspace" on page 67.

Creating Child Workspaces

After you create a parent workspace, team members will need to create their own child workspace with copies of the parent workspace files. Configuring transactions revolve around these parent-child relationships: bring over files from the parent workspace; change files in the child workspace; put back files to the parent workspace.

To create a child workspace:

1. **Start Configuring.**
2. **If the workspace from which you must obtain your files is not automatically loaded, choose File ➤ Load Workspaces.**

3. **Select the workspace in the Load Workspaces dialog box and click Load Workspaces.**

   The parent workspace is loaded and its icon appears in the Configuring window.

4. **Choose Actions ➤ Bringover Create.**

   This opens the Bringover Create Tab in the Transactions dialog box (see FIGURE 2-2).

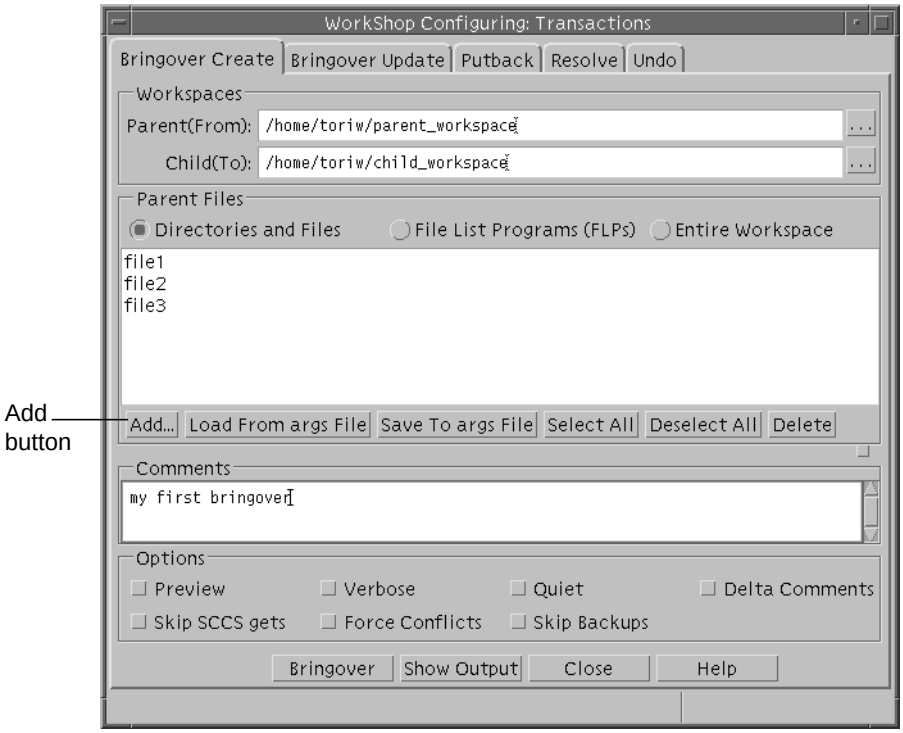


FIGURE 2-2 Transactions Dialog Box: Bringover Create Tab

5. **To select the directories and files to bring over, do one of the following:**
   - Accept the default "`./`" to bring over all files
   - Click the Entire Workspace button to bring over all files
   - Click the Add button to display the Add Files dialog box where you can add or delete specific files.

   a. **In the Add Files dialog box** (FIGURE 2-3)**, navigate to the files you want to include.**

      Navigate through the file system hierarchy by double-clicking on any directory icon. Double-click on the directory icon to move hierarchically upward in the file system. To move directly to a directory, type its path name in the Name text box and click the Load Directory button. You cannot move outside of the parent workspace hierarchy.

FIGURE 2-3 Add Files Dialog Box

b. Select files and directories.

Click any file or directory icon. You can Shift-click to select multiple files and directories.

c. Click Add Files to add the files to the Bringover Create tab.

d. Click Close.

6. In the Bringover Create tab, click the Bringover button.

A Transaction Output window displays the status of the bringover and indicates when the bringover is complete.

You now have a child workspace from which you can check out files, make changes to them, and put back to the parent workspace. To learn how to check out files, see "Checking Out a File" on page 67. To learn how to put back files to a parent workspace, see "Propagating Changes Across Workspaces" on page 25.

Propagating Changes Across Workspaces

After you have created a hierarchy of workspaces, it is important to keep the contents synchronized. All Configuring transactions are performed from the perspective of the child workspace; hence the Bringover transaction "brings over" groups of changes from the parent to the child workspace. Inversely, the Putback transaction "puts back" changes from the child workspace to its parent.

Use the Bringover Update transaction to update changes from the parent workspace to a child workspace. Use the Putback transaction to take changes in a child workspace and put them in a parent workspace. Putting the files back into the parent makes those changes available to other members of the team.

Updating a Child Workspace (Bringover Update)

To initiate a Bringover Update transaction:

1. **In Configuring, load your workspace with File ➤ Load Workspace.**
2. **Click on your workspace to select it.**
3. **Choose Actions ➤ Bringover Update.**

   This displays the Bringover Update tab of the Transactions dialog box (see FIGURE 2-4).

   The parent and child names are automatically inserted in the Workspaces text boxes. You can insert a new path name, and edit the text box at any point.

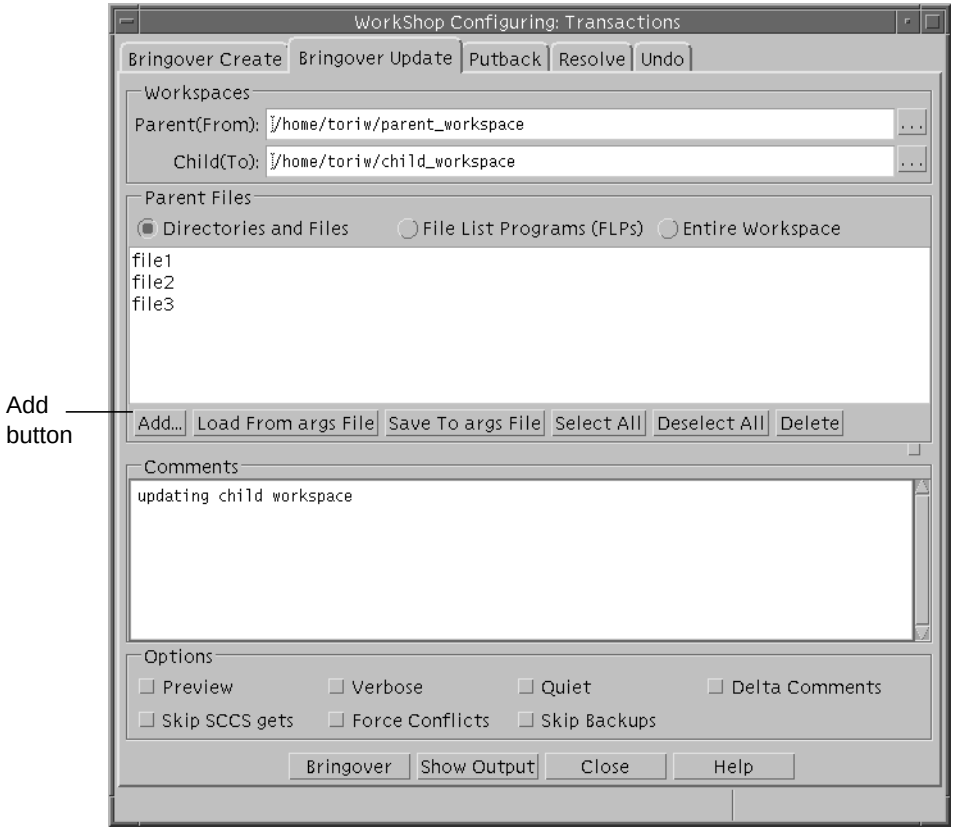


FIGURE 2-4 Transactions Dialog Box: Bringover Update Tab

4. **To select the directories and files to bring over, do one of the following:**
   - Accept the default "`./`" to bring over all files
   - Click the Entire Workspace button to bring over all files
   - Click the Add button to display the Add Files dialog box where you can add or delete specific files.

   a. **In the Add Files dialog box** (see FIGURE 2-3)**, navigate to the files you want to include.**

      Navigate through the file system hierarchy by double-clicking on any directory icon. Double-click on the directory icon to move hierarchically upward in the file system. To move directly to a directory, type its path name in the Name text box and click the Load Directory button. You cannot move outside of the parent workspace hierarchy.

b. Select files and directories.

Click any file or directory icon. You can Shift-click to select multiple files and directories.

c. Click Add Files to add the files to the Bringover Update tab.

d. Click Close.

5. **In the Bringover Update tab, click the Bringover button.**

A Transaction Output window displays the status of the bringover and indicates when the bringover is complete. You now have the most recent copies of the files from the parent workspace in your child workspace.

Note – If you want to preview your transaction, click the Preview option to verify your transaction before you transfer any files.

Putting Back Changes to a Parent Workspace

To initiate a Putback transaction:

1. **In Configuring, load your workspace with File ➤ Load Workspace.**
2. **Click on your workspace to select it.**
3. **Choose Actions ➤ Putback.**

This displays the Putback tab of the Transactions dialog box (see FIGURE 2-5).

The names are automatically inserted in the Workspaces text boxes. You can insert a new path name and edit the text box at any point.

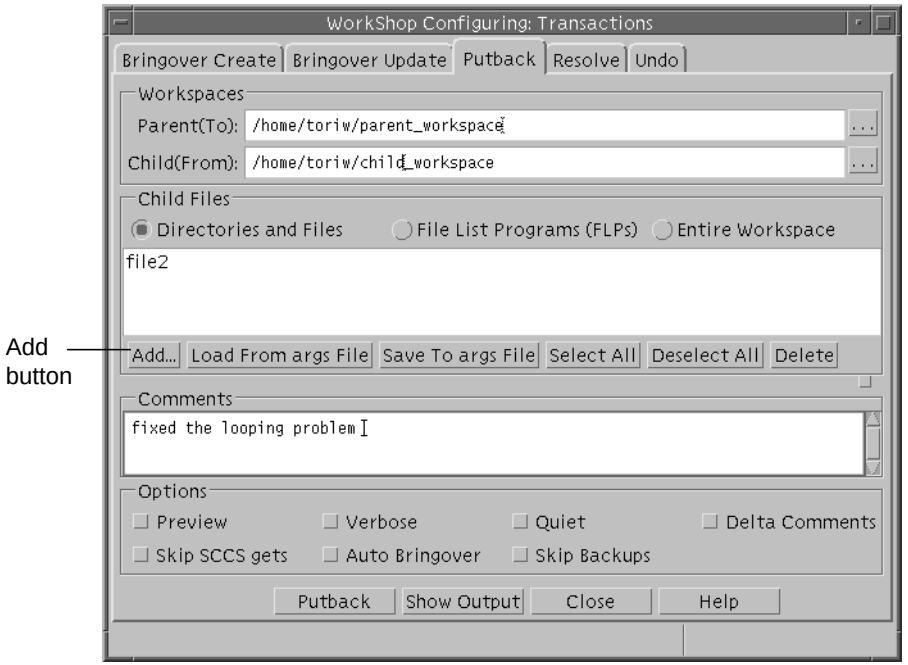


FIGURE 2-5 Transactions Dialog Box: Putback Tab

4. **To select the directories and files to put back, do one of the following:**
   - Accept the default "`./`" to bring over all files
   - Click the Entire Workspace button to bring over all files
   - Click the Add button to display the Add Files dialog box where you can add or delete specific files.

   a. **In the Add Files dialog box** (see FIGURE 2-3)**, navigate to the files you want to include.**

      Navigate through the file system hierarchy by double-clicking on any directory icon. Double-click on the directory icon to move hierarchically upward in the file system. To move directly to a directory, type its path name in the Name text box and click the Load Directory button. You cannot move outside of the parent workspace hierarchy.

   b. **Select files and directories.**

      Click any file or directory icon. You can Shift-click to select multiple files and directories (see FIGURE 2-3).

   c. **Click Add Files to add the files to the Putback tab in the Transactions dialog box (**FIGURE 2-5**).**

   d. **Click Close.**

5. **Type a comment that describes the Putback transaction.**

   This comment is included in the Workspace History and can be up to 8 Kilobytes long.

6. **In the Putback tab, click the Putback button.**

   A Transaction Output window displays the status of the putback and indicates when the putback has completed. You now have synchronized the contents of your files and the files in the parent workspace.

Note – When you try to perform a putback on a file that you have changed in your child workspace, but which has already been changed in the parent workspace, Configuring prevents you from putting back the file until you have resolved the differences between them. See Chapter 6.

Action taken during the Putback transaction can be reversed using the Undo transaction. Refer to the next section, "Undoing Changes to a Workspace," for details.

Undoing Changes to a Workspace

You can reverse (undo) the action of the most recent Bringover or Putback transaction in a workspace by using the Undo tab in the Transactions dialog box. 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 want until another Bringover or Putback transaction occurs in that workspace; only the *most recent* Bringover or Putback transaction can be undone.

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

To initiate an Undo transaction:

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 box. At any point, you can insert a new path name followed by a Return, or change the text box.

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

Renaming or Moving Workspaces

Configuring does all the administrative work of keeping track of workspaces, files and their relationships. It is important that you use Configuring to rename, move or delete workspaces so it can maintain workspace histories and the relationships between workspaces.

Note – Use these procedures to rename or move workspaces, rather than with the the Common Desktop Environment (CDE) FileManager or the SunOS™ operating system command `mv`. Using the procedures detailed below will maintain the parent-child relationships of workspaces.

Rename or move workspaces with the move command available from the Configuring Workspace menu.

To rename or move a workspace:

1. **Load the workspace with File ➤ Load Workspaces.**
2. **Select the workspace by clicking on it once.**
3. **Choose Workspace ➤ Rename.**
4. **In the Rename dialog box, type in the new name or location for your workspace.**
5. **Click OK.**

   The workspace with the new name or location appears in the Configuring window.

Deleting or Reconverting Workspaces

Use these procedures to delete or reconvert workspaces, rather than using Operating System commands.

Deleting a Workspace

To delete a workspace:

1. **Load the workspace with File ➤ Load Workspaces.**

2. **Select the workspace by clicking on it once.**

   You can Shift-click to select multiple workspaces.

3. **Choose Workspace ➤ Delete.**

4. **Click OK in the Delete Confirmation dialog box.**

   The workspace icon disappears in the Configuring window.

Reconverting a Workspace

To convert a Sun WorkShop TeamWare workspace back into a regular directory:

1. **Load the workspace with File ➤ Load Workspaces**

2. **Select the workspace by clicking on it once.**

   You can Shift-click to select multiple workspaces.

3. **Choose Workspace ➤ Delete.**

4. **Select the Delete Codemgr_wsdata Directory Only button.**

5. **Click OK in the Delete Confirmation dialog box.**

   The workspace icon disappears in the Configuring window, but the files remain intact.

Viewing Workspace History

Configuring transactions are logged in the workspace history file.

- Commands that affect a single workspace are logged only in that workspace.
- Commands that affect more than one workspace are logged in both the source and destination workspaces. Although command entries are logged in both the source and destination workspaces, the list of changed files is entered only in the destination directory.

You can view the contents of the workspace history file to track or reconstruct changes that have been made to a workspace over time. Command log entries consist of the underlying command-line entries. If you have any questions about the meaning or syntax of a command, refer to its `man` page for details. For information about accessing man pages, see Chapter 11.

To view the history of a workspace:

1. **From the Configuring Window, click on a workspace to select it.**
2. **Choose Workspace ➤ View History.**

   The Workspace History Viewer opens (see FIGURE 2-6).

Changing the Workspace History Viewer Display

You can customize the following in the Workspace History Viewer:

- How much transaction information is included
- In what order the transactions appear
- Which items are included

By default, the status of the transaction is the only transaction information displayed in the Workspace History Viewer. TABLE 2-2 shows the categories for the Workspace History Viewer Display.

TABLE 2-2 Workspace History Viewer Display Categories

Category	Contents
Time	Date and time of the transaction
Operation	Transaction name
User	User that initiated the transaction
Host	Name of the machine from which the transaction was initiated
TW Release	Which version of TeamWare executed the transaction
Status	Exit status of the transaction

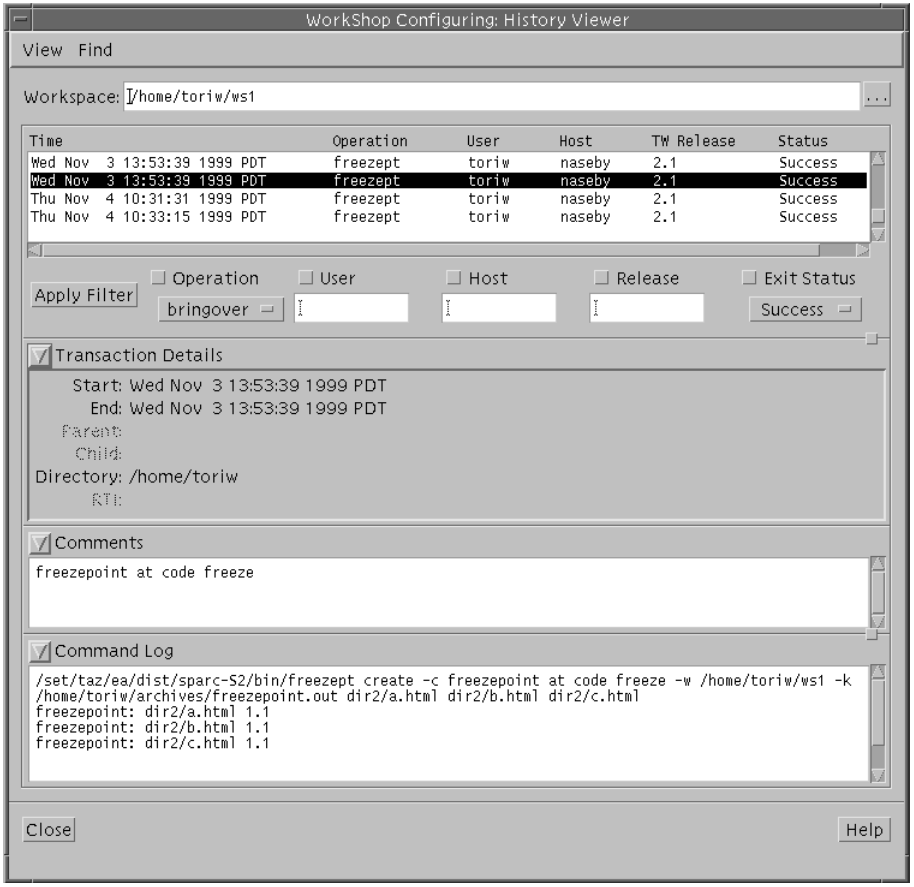


FIGURE 2-6 Workspace History Viewer

Adding or Removing Transaction Information From the Workspace History Viewer

To add or remove transaction information:

1. **In the Workspace History Viewer Window, choose View ➤ Options.**
2. **In the Workspace History Viewer Options dialog box, check the items in the Display section that you want included in or removed from the Workspace History report.**
3. **Click OK.**

   The Workspace History Viewer Options dialog box disappears, and the details you have selected are included in the Workspace History Viewer window.

Sorting Transaction Information in the Workspace History Viewer

By default, transactions are sorted chronologically, with the most recent transactions at the bottom.

1. **In the Workspace History Viewer Window, choose View ➤ Options.**
2. **In the Workspace History Viewer Options dialog box, click one of the Sort By radio buttons.**
3. **Click OK.**

   The transaction information is sorted based on the category you selected.

Adding Filtering Options to the Workspace History Viewer

You can filter the Workspace History transaction information by Operation, User, Host, Release, or Exit Status. By default, all transactions appear in the Transaction Information pane and the only filter option displayed is User. If the category you want to use to filter transactions does not appear in the Workspace History Viewer, you must add the option.

To add filter options to the Workspace History Viewer window:

1. **In the Workspace History Viewer Window, choose View ➤ Options.**
2. **In the Options dialog box, check the items in the Filter section that you want to use as a filter.**
3. **Click OK.**

   The items you selected appear in the Workspace History Viewer window.

Filtering Transaction Information From the Workspace History Viewer

To filter transaction information:

1. **In the Workspace History Viewer Window, choose View ➤ Options.**
2. **Click the filter check box above the filter(s) you want to use: click Operation, User, Host, Release or Exit Status.**

   You can select more than one filter.
3. **Type the user, host, or release you want to filter by, or select an Operation or Exit Status.**
4. **Click Apply Filter.**

Searching for Transactions

If you want to find a particular transaction, you can search for text strings in the comments or command log.

To search the Comments:

1. **In the Workspace History Viewer, choose Find ➤ Search Comments.**
2. **In the Search Comments dialog box, type a search string.**
3. **Click Find Next.**

   If Configuring finds the search string, it will highlight it in the History Viewer Comments pane.

To search the Command Log:

1. **In the Workspace History Viewer, choose Find ➤ Search Command Log.**
2. **In the Search Command Log dialog box, type a search string.**
3. **Click Find Next.**

   If Configuring finds the search string, it will highlight it in the History Viewer Command Log pane.

CHAPTER 3

Advanced Workspace Management

To learn basic Configuring tasks, see Chapter 2. This chapter covers these advanced Configuring tasks:

- Using Bringover/Putback Options
- Creating Customized Bringover/Putback File Lists
- Notifying Users of Transactions
- Giving a Workspace a Descriptive Name
- Reparenting a Workspace
- Customizing Configuring Using Tool Properties
- Configuring Environment Variables
- Converting From an RCS Project

Using Bringover/Putback Options

There are two types of options for bringovers/putbacks:

- Options in the Transactions dialog box that are available to you each time you perform a bringover/putback
- Tool properties that you set for all bringovers/putbacks

Setting Options During a Bringover/Putback

To access options during a bringover/putback:

1. **In the Configuring window, choose a transaction from the Action Menu:**
   - Bringover Create
   - Bringover Update
   - Putback
2. **Click a check box in the Options section (see** FIGURE 3-1**).**

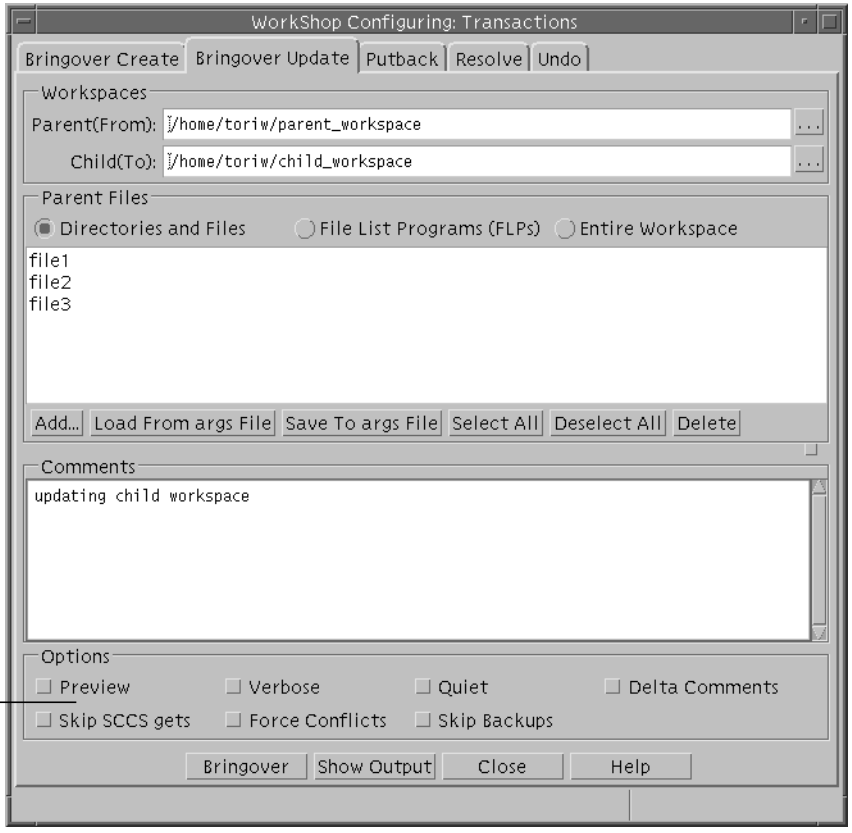


FIGURE 3-1 Transactions Dialog Box: Options Section

Each check box in the Options section of the Bringover Create/Bringover Update/ Putback tabs in the Transactions dialog box is described in TABLE 3-1.

TABLE 3-1 Bringover/Putback Options Check Boxes

Check Box	Description
Preview	Previews the results of the transaction. If you invoke a transaction with this option selected, the transaction proceeds without transferring any files. You can monitor the output messages in the Transaction Output window and verify the expected outcome of the transaction.
Verbose	Increases the amount of information displayed in the Transaction Output window. By default, a message is displayed for each created, updated, or conflicting file. Verbose causes Configuring to print a message for each file, including those that are not brought over. If both the Verbose and the Quiet are specified, the Quiet option takes precedence.
Quiet	Suppresses the output of status messages to the Transaction Output window.
Delta Comments	Includes additional information in the Transaction Output window: delta number, owner, and comments.
Skip SCCS `gets`	Inhibits the automatic invocation of the SCCS `get` command as part of the Bringover transaction. Normally g-files are extracted from the SCCS history after they are brought over. This option improves transaction performance although it shifts the responsibility to the user to do the appropriate `gets` at a later time.
Force Conflicts	Cause all updates to be treated as conflicts.
Skip Backups	Skips the step of copying the existing files to the `Codemgr_wsdata/backup/files` directory in the destination workspace. This option reduces the disk space occupied by the child workspace and improves transaction performance, but it removes the option of using Undo.

Setting Tool Property Options

Use the Bringover/Putback Options tab to set options for bringovers and putbacks. To open the Bringover/Putback Options tab, in the Configuring Window:

1. **Choose Options ➤ Configuring.**
2. **Select the Bringover/Putback tab.**

TABLE 3-2 lists the check boxes for the Bringover/Putback Tool Properties.

TABLE 3-2 Bringover/Putback Tool Properties

Check Box	Description
Transaction File List: Autoload	Reads the `Codemgr_wsdata/args` file and loads it into the File Pane whenever a new workspace is selected. Deselect this property when you want to use the same file list for a number of transactions involving different workspaces. The default is on.
Transaction Output: Auto Display	Displays the Transaction Output dialog box during Bringover, Putback, and Undo transactions. If this is not selected, you must activate the Transaction Output dialog box using the Show Output button. The default is on.
Putback: Auto Bringover Update	Invokes a bringover when a putback determines that files have changed in the parent. The default is off.
Bringover/Putback: Warn Comment Reusing	Displays an alert reminding you that the comments are the same as the last Bringover/Putback. The default is on.

Creating Customized Bringover/ Putback File Lists

Configuring maintains a list of files each time you bring over or put back to a specific workspace. This section explains how Configuring generates a default list of files for bringovers/putbacks and how you can change the default. You can also direct Configuring to generate a list of files to bringover or putback using an FLP (File List Program).

Saving a Default List of Files

Configuring saves the list of files and directories you include when performing a bringover or putback. This list is loaded by default for the next bringover/putback transaction in the same workspace. Each time you perform a bringover/putback, Configuring determines whether the file list is more encompassing than the list from the previous transaction; if the new list is of a wider scope, the new list replaces the old.

You can reload the default list at any time by clicking Load from args File. This is useful if you have made changes to the list that you do not want to save. Use Load List from args File to revert the list to its default state.

If you change the list and want to make the new list the default, click Save to args File. This is useful if you have eliminated files or directories from the list. If you add files, Configuring automatically adds them to the `args` file as part of a Bringover or Putback transaction.

Generating a Customized List of Files

In addition to explicitly specifying individual files for transfer, you can direct Configuring to execute a program to create a customized list of files to bringover/putback.

By default, Sun WorkShop TeamWare generates a list of files to include in a bringover/putback transaction by using *File List Program* (*FLP*). The FLP generates a list of files and then passes this list to bringover and putback transactions. Configuring uses 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. The files generated by this (or any) FLP are included for transfer, in addition to any *files* that you also specify in the File List pane.

If you want to have more control over which files are included in a bringover or putback, you can write your own FLP. You can, for example, include only files with a certain extension, such as html. If you want to use your own FLP(s) during a transaction, you specify them in the File List pane.

To generate your own list of files for a bringover/putback:

1. **Write an File List Program (FLP) that creates the list of files you want to bringover/putback.**

   For example:

   ```
   # my FLP
   cd subdir/webfiles
   ls *.html
   ```

2. **In the Bringover/Putback tab, click the File List Programs (FLPs) button.**
3. **Click Add.**
4. **Select your FLP in the Add FLPs dialog box.**
5. **Click Add Files.**

This sets the FLP for the current transaction only. You can direct Configuring to always use your FLP by setting the `CODEMGR_DIR_FLP` environment variable. This variable overrides Configuring's default FLP, which is named `def.dir.flp`.

```
% setenv CODEMGR_DIR_FLP /home/workspaces/my.flp
```

If you use the bringover or putback commands at the command line, use the `-f` option to specify an FLP. For more information about using command line commands, see Chapter 11.

Notifying Users of Transactions

You can have Configuring automatically send out an email to members of your team each time a transaction occurs in a workspace. The email will contain the transaction type, file names, and transaction comments.

To set up email notification:

1. **Start Configuring.**
2. **Select Workspace ➤ Properties.**
3. **In the Workspace Properties dialog box, type the name of a workspace.**
4. **Click the Notification tab.**
5. **In the Notification tab, click Create Entry.**
6. **In the Notification Entry dialog box:**
   a. **Type an email address in the Mail To text box.**
   b. **Select the transactions for which you want to generate an email notification.**
   c. **If you want to generate a notification only for certain files in the workspace, click the Specify button.**
      - Click the Add to List button and select the files you want to include.
      - Click Add files.
      - Click Cancel to close the Add Files dialog box.
   d. **Click OK in the Notification Entry dialog box.**
7. **To save the entry, click OK in the Workspace Properties dialog box.**

Giving a Workspace a Descriptive Name

Often, a workspace name is a long path, such as `/home/src/rel7/ver2`. Sun WorkShop TeamWare lets you give a workspace a name that is more meaningful to users, such as "current development area" or "Bob's workspace." The descriptive name is displayed in the Configuring window (when you select View ➤ Descriptive names), and appears as a part of email notification.

You can also create a detailed description of the workspace for your own use. The detailed description is displayed when you select View ➤ Descriptive names.

To give a workspace a descriptive name:

1. **Start Configuring.**
2. **Choose File ➤ Load Workspaces.**
3. **Load the workspace you want to give a descriptive name.**
4. **Click on the workspace to select it.**
5. **Choose Workspace ➤ Properties.**
6. **In the Workspace Properties Dialog Box, click the Description tab.**
7. **In the Name text box, type a descriptive name for your workspace.**

   This name appears as a label for the workspace.

8. **In the Description text box, type a longer description of your workspace.**

   This information is stored in the `Codemgr_wsdata/description` file.

Note – Clicking Load in the Description tab displays description information from the description file. It does not save any information.

---

9. **Click OK.**

   To view descriptive names:

   - In the Configuring window, choose View ➤ Descriptive Names.
   - Use the `workspace descr` command.

   The syntax is:

```
workspace descr [ -n | -d | -a ] wsname ...
```

TABLE 3-3 lists the options to the `workspace descr` command.

TABLE 3-3 `workspace descr` Command Options

Option	Description
`-n`	Lists descriptive name only
`-d`	Lists detailed description only
`-a`	Lists both descriptive name and detailed description (default)
`wsname`	Workspace name

For more information about using command line commands, see Chapter 11.

Reparenting a Workspace

As discussed in "Parent and Child Workspaces" on page 11, the parent-child relationship is the thread that connects workspaces to each other. Configuring provides the means for you to change this relationship. This section discusses:

- Reasons to Change a Workspace's Parent
- Ways to Reparent Workspaces
- A Reparenting Example

Reasons to Change a Workspace's Parent

You can permanently or temporarily change a workspace parent for any of these reasons:

- To populate a new top-level workspace. You have just completed Release 1 of your product and need to begin work on Release 2. You can:

1. Create a new (empty) Release 2 workspace (File ➤ Create Workspace).
2. Make the Release 2 workspace the new parent of the Release 1 workspace.
3. Use the Putback transaction to copy files to the Release 2 workspace.
4. 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.

- 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. Use the Configuring Putback transaction to incorporate the changes into the new parent. An example of this use for reparenting is included in "A Reparenting Example" on page 46.
- To reorganize workspace hierarchies. You can:
  - Add additional levels to the hierarchy
  - Remove levels from the hierarchy (do not specify a new parent during reparenting)
  - Reorganize workspace branches within the project hierarchy
- To adopt an orphan workspace. If a file is orphaned (for example, if its `Codemgr_wsdata/parent` file has been deleted or its parent is corrupted), you can use the reparenting feature to restore its parentage.

Ways to Reparent Workspaces

There are two equivalent ways to reparent workspaces permanently: using the Reparent command or dragging and dropping workspace icons. You can also change a workspace's parent during a bringover or putback transaction, but this new relationship lasts only for the duration of the transaction.

The Reparent Command

To reparent a workspace using the reparent command:

1. **Start Configuring.**
2. **Choose File ➤ Load Workspaces.**
3. **Load the workspace you want to reparent.**
4. **Click on the workspace to select it.**
5. **Choose Workspace ➤ Reparent.**
6. **Type the name of the new parent in the Reparent dialog box.**
7. **Click OK.**

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

Drag-and-Drop Workspace Icons

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

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

Temporary Reparenting

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 in the Bringover Update tab (Actions ➤ Bringover Update). You can also 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 box in the Putback tab (Actions ➤ Putback). You change the parent for that transaction only.

A Reparenting Example

When a bug is fixed in a version of a product, often a patch release is made to distribute the fixed code. The code that was fixed must also be incorporated into the next release of the product as well. If the product is developed using Sun WorkShop TeamWare, 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 the Release 2.0 code, which has already begun development.

1. **In the Configuring window, you load two workspaces, `Release2.0` and `patch1.0`.**

   These workspaces do not have a parent-child relationship (see FIGURE 3-2).

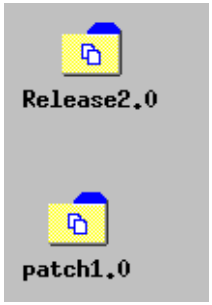


FIGURE 3-2 Two Unrelated Workspaces

2. **Before you make any changes, make a child of the `patch1.0` workspace using a Bringover Create transaction. (see** FIGURE 3-3**)**

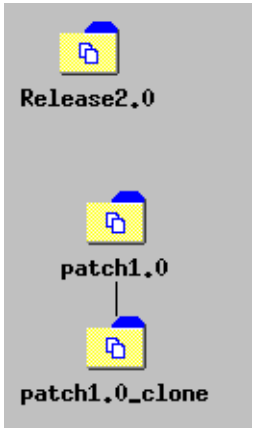


FIGURE 3-3 Clone Workspace Created

3. **Change the parent of `patch1.0_clone` from `patch1.0` to `Release2.0` using the reparent command (see** FIGURE 3-4**).**

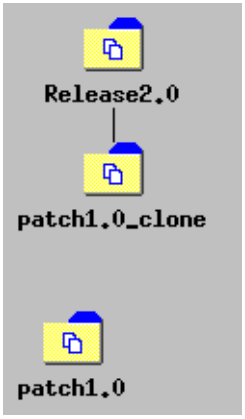


FIGURE 3-4 Clone Workspace Reparented to Release2.0

4. **Update the `patch1.0_clone` with a Bringover Update from its new parent, `Release2.0`** (see FIGURE 3-5).

   This includes merging the fixes made for the patch in `patch1.0_clone` with the files from `Release2.0`.

5. **Put back the changes from `patch1.0_clone` to `Release2.0`** (FIGURE 3-5).

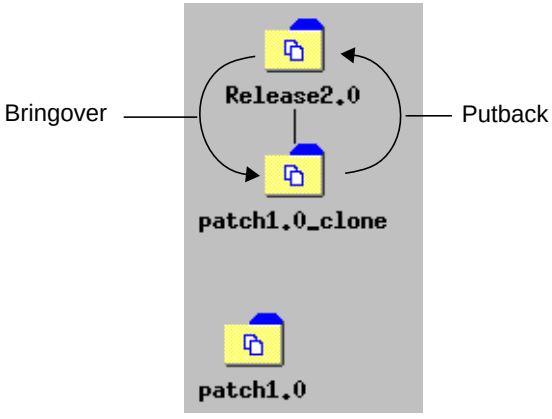


FIGURE 3-5 Files Brought Over, Merged, and Incorporated into the New Release

6. **Now that it has served its purpose, you can delete `patch1.0_clone` using Workspaces ➤ Delete.**

You now have the two unrelated workspaces, `Release2.0`,which now includes the fixes from `patch1.0` and `patch1.0`,which is unchanged from the transactions. Now the patch is available to the children of `Release2.0` (see FIGURE 3-6).

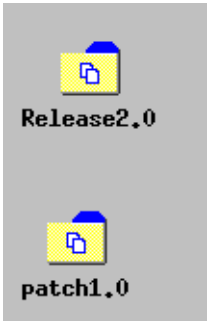


FIGURE 3-6 `patch1.0_clone` Deleted; `Release2.0` Includes Fixes

Customizing Configuring Using Tool Properties

Using the Tool Properties dialog box (see FIGURE 3-7), you can customize the behavior of:

- Configuring window functions
- Bringover/Putback transactions
- Resolve transaction
- Workspace History Viewer

To open the Tool Properties dialog box, choose Options ➤ Configuring. The tabs in the dialog box lets you switch between the Configuring, Bringover/Putback, Resolve, and Workspace History Viewer panes. Options in the Resolve tab are described in “Merging Options” on page 80.

The CodeManager tab of the Tool Properties dialog box (see FIGURE 3-7) lets you change the behavior of the Configuring main window. The specific properties are described in TABLE 3-4.

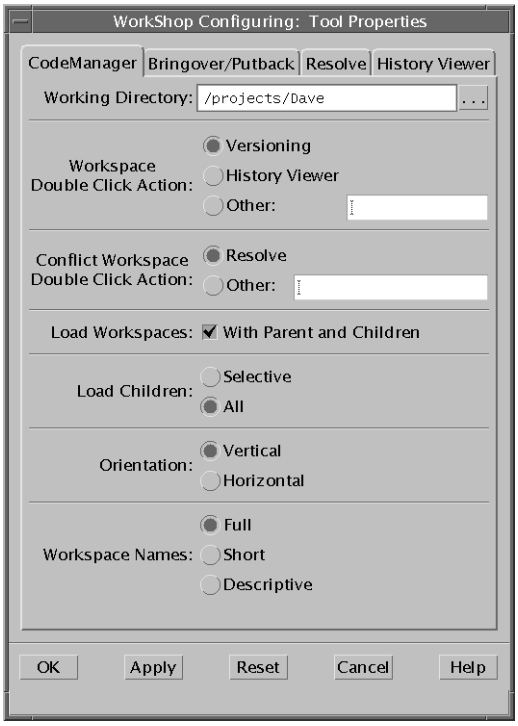


FIGURE 3-7 Tool Properties: CodeManager Tab

TABLE 3-4 Configuring Tool Properties

Property	Description
Working Directory	Lets you specify the directory to which Configuring actions are relative.
Workspace Double Click Action	Lets you specify the commands you want launched when you double-click on a standard workspace icon. Select Versioning, History Viewer, or Other and type the path name of a command. The command executes based on the working directory and your search path. The default double-click action is Versioning (`twversion`).
Conflict Workspace Double Click Action	Lets you specify the commands you want launched when you double-click on workspaces that contain conflicts. Type the path name required to execute the commands based on the working directory and your search path. By default, the Resolve Transaction window opens 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.

TABLE 3-4 Configuring Tool Properties *(Continued)*

Property	Description	
Load Children	These radio buttons are active only if you have the Load Workspaces check box checked. Select Selective if you want to specify which child workspaces to load. Select All if you want all children to be loaded. By default, all children are loaded.	
Orientation	Select Vertical if you want workspace hierarchy displayed vertically from top to bottom. Select Horizontal if you want the workspace hierarchy displayed horizontally from left to right in the Workspace Graph pane. Vertical is the default. This property corresponds to the choosing View ➤ Orientation the Configuring main window.	
Workspace Names	Changes the format of workspace names. This property corresponds to choosing View ➤ Names in the Configuring main window.	
	Full	Displays workspaces labeled with full path names in the Workspace Graph pane. (Default.)
	Short	Displays workspaces labeled with just the file name.
	Descriptive	Displays the descriptive name you have assigned to a workspace. See "Giving a Workspace a Descriptive Name" on page 43. If there is no descriptive name for the workspace, and you select Descriptive name, the file name appears in angled brackets, for example `<myworkspace>`.

Configuring Environment Variables

This section provides examples of how to use Configuring environment variables:

- Loading Workspaces Automatically
- Setting Focus for Command-Line Commands
- Setting a Search Path

For information about how to set a default FLP using the `CODEMGR_DIR_FLP` variable, see "Generating a Customized List of Files" on page 41.

Loading Workspaces Automatically

You can set the CODEMGR_WSPATH variable to a single workspace, a list of workspaces, or to all the workspaces in a directory. To set the CODEMGR_WSPATH variable to the location of the workspace, type:

```
% setenv CODEMGR_WSPATH /home/ws/myworkspace
```

To load more than one workspace, put a list of workspaces in quotes:

```
% setenv CODEMGR_WSPATH "/home/ws/myworkspace /home/ws/anotherws"
```

To load all the workspaces in one directory, set the variable to a directory that contains multiple workspaces. For example:

```
% setenv CODEMGR_WSPATH /home/ws/myworkspaces
```

Setting Focus for Command-Line Commands

The CODEMGR_WS environment variable sets a workspace as the default for command-line commands. Once you set this variable, when you use command-line commands (bringover, putback, freezept extract, and so on) the workspace will be used automatically if you don't specify a workspace with the -w option. To set a default workspace:

```
% setenv CODEMGR_WS /home/workspaces/myworkspace
```

This variable also has the effect of loading the workspaces when you start Sun WorkShop TeamWare tools.

For more information about command-line commands, see Chapter 11.

Setting a Search Path

The CODEMGR_PATH_ONLY environment variable lets you dictate where Sun WorkShop TeamWare tools look for other Sun WorkShop TeamWare tools. To set the CODEMGR_PATH_ONLY environment variable:

```
% setenv CODEMGR_PATH_ONLY /bin/install/Teamware
```

If the CODEMGR_PATH_ONLY variable is not set, Sun WorkShop TeamWare looks for other tools in the current directory the tool is in, and then searches in the directories specified in the PATH environment variable.

Converting From an RCS Project

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

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

To convert files, rcs2ws invokes the RCS co command and the SCCS admin, get, and delta commands. rcs2ws finds these commands using your PATH variable. If rcs2ws cannot find the SCCS commands, it looks for them in the /usr/ccs/bin directory.

Note – rcs2ws requires that you have the RCS utility. If you get the error "command not found," make sure you have RCS and that the location of RCS is set in your PATH.

---

rcs2ws does not convert RCS keywords to SCCS keywords. Keywords are treated as text in the SCCS delta.

The basic syntax of rcs2ws is:

```
rcs2ws -p [RCS_source_dir] -w [teamware_workspace] [files | directory]
```

The `-p` option names the RCS source directory and is required. Relative file names are interpreted as being relative to *RCS_source_dir*.

The `-w` option names the TeamWare workspace. If the workspace does not exist, using the `-w` option will create a workspace. The `-w` is optional if the workspace already exists and it is your default workspace, or if the current directory is contained within an existing TeamWare workspace.

For example, if you want to convert the RCS project `/projects/prodA/release1` into a new TeamWare workspace `/tw/workspaces/dev1`, type:

```
% rcs2ws -p /projects/prodA -w /tw/workspaces/dev1 release1
```

If the workspace `/tw/workspaces/dev1` already exists and it is your default workspace, you could type:

```
% rcs2ws -p /projects/prodA release1
```

Use "." to specify that every RCS file under *RCS_source_dir* should be converted. For example, if you want to convert all the RCS files in the project directory `/projects/prodA`, type:

```
% rcs2ws -p /projects/prodA .
```

To see a complete list of `rcs2ws` options, see the `rcs2ws(1)` man page.

CHAPTER 4

Controlling Workspace Access

Sun WorkShop TeamWare allows you to control who can putback files to workspaces by setting permissions and through putback validation. This chapter covers the following topics:

- Specifying Permissions
- Protecting Workspaces With Putback Validation
- Removing Workspace Locks

Specifying Permissions

You can grant or deny permissions on a global (see "Specifying Global Permissions" on page 56), group, or individual basis (see "Specifying Group or Individual Permissions" on page 57).

You can grant or deny permission for individual users or groups to perform the following Sun WorkShop TeamWare transactions:

- Bringover-from
- Bringover-to
- Putback-from
- Putback-to
- Undo
- Workspace delete
- Workspace move
- Workspace reparent
- Workspace reparent-to

Note – Transactions can take longer to complete when you set access control. If you grant or deny permissions to a large group, Configuring can take several seconds to look up the members of the group before executing a transaction.

Specifying Global Permissions

To specify global permissions:

1. **Start Configuring.**
2. **Choose File ➤ Load Workspaces.**
3. **Select the workspace in the Load Workspaces dialog box and click Load Workspaces.**
4. **Click on the workspace to select it.**
5. **Choose Workspace ➤ Properties.**
6. **Select the Access Control tab in the Workspace Properties dialog box.**

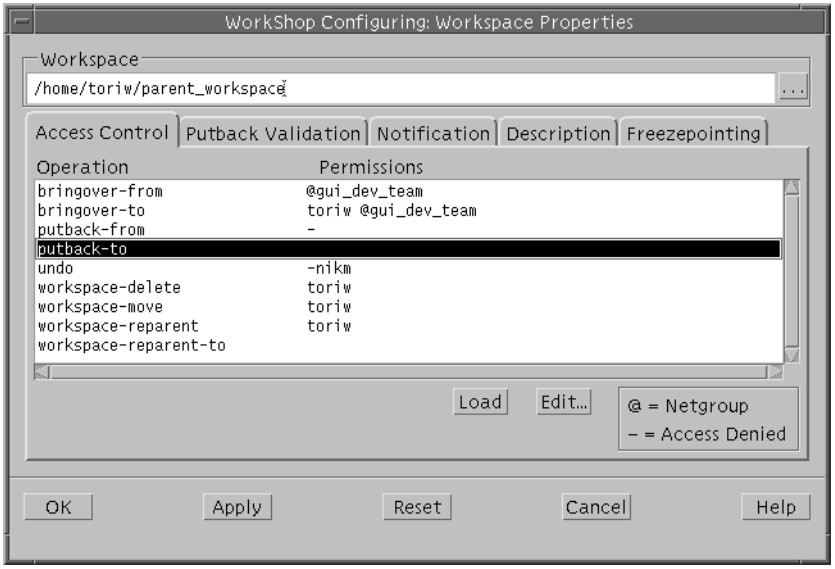


FIGURE 4-1 Workspace Properties: Access Control Tab

7. **Select an operation and click Edit.**

   This displays the Access Control Properties dialog box.

8. **Select the type of permission you want to allow:**

   - None: No users have permission
   - All: All users have permission

   By default, all users have permission to execute the transaction.

9. **Click OK.**

Specifying Group or Individual Permissions

1. **Start Configuring.**
2. **Choose File ➤ Load Workspaces.**
3. **Select the workspace in the Load Workspaces dialog box and click Load Workspaces.**
4. **Click on the workspace to select it.**
5. **Choose Workspace ➤ Properties.**
6. **Select the Access Control tab in the Workspace Properties dialog box.**
7. **Select an operation and click Edit to display the Access Control Properties dialog box.**
8. **Click the Specify permissions radio button.**
9. **In the Name text box, type a user's (or group's) login name.**
10. **Click the User or Netgroup button to indicate whether it is a single user or a group.**
11. **Click the Granted or Denied radio button.**
12. **Click the Insert Before or Insert After button to add the user to the list (or to start the list).**
13. **Click OK.**

    The user or netgroup is displayed on the permissions list in the Access Control tab.

Note – The order in which a user appears on the list can have an effect on permissions. If a user is listed as having been both granted and denied permission, Sun WorkShop TeamWare tools use only the first reference. This can occur when you grant access to an individual and deny access to a group that the individual belongs to. In this instance, if the individual is listed first, the individual will have access, if the group is listed first, the individual does not have access.

Protecting Workspaces With Putback Validation

Use putback validation if you want to have more strict control over a workspace. When you turn on putback validation, only putbacks are allowed to the workspace. The user is also prompted for a "password" (Integration Request Identifier) before doing a putback. Configuring does not validate the Integration Request Identifier, but it will pass it on to another program. To validate the Integration Request Identifier, you must write your own validation program. See "Invoking Your Own Putback Validation Program" on page 59. If Putback Validation is turned on, but you do not have your own putback validation script, Configuring always allows the putback. The Integration Request Identifier supplied by the user is recorded in workspace history file and under the heading RTI (Request To Integrate).

TABLE 4-1 summarizes the three Putback Validation modes.

TABLE 4-1 Putback Validation Modes

Mode	User	Configuring
Putback Validation off (default)	Does nothing.	Allows all putbacks to workspace.
Putback Validation on	Turns on Putback Validation.	Requires a "password" (Integration Request Identifier) before a putback is allowed. Password is not validated, merely recorded.
Putback Validation on and passwords checked.	Writes putback validation program, turns on Putback Validation, invokes putback validation program.	Requires a "password" (Integration Request Identifier) before a putback is allowed. Integration Request Identifier is passed to validation program. Validation program must grant permission before putback is allowed.

SCCS also allows you to require a check-in validation for individual files. See "Setting SCCS File Properties" on page 102.

Turning On Putback Validation

To turn on Putback Validation:

1. **In the Configuring Window, load an existing workspace by choosing File ➤ Load Workspaces.**
2. **Click on the workspace to select it.**
3. **Choose Workspace ➤ Workspace Properties.**
4. **Click the Putback Validation tab in the Workspace Properties Dialog Box (see** FIGURE 4-2**).**

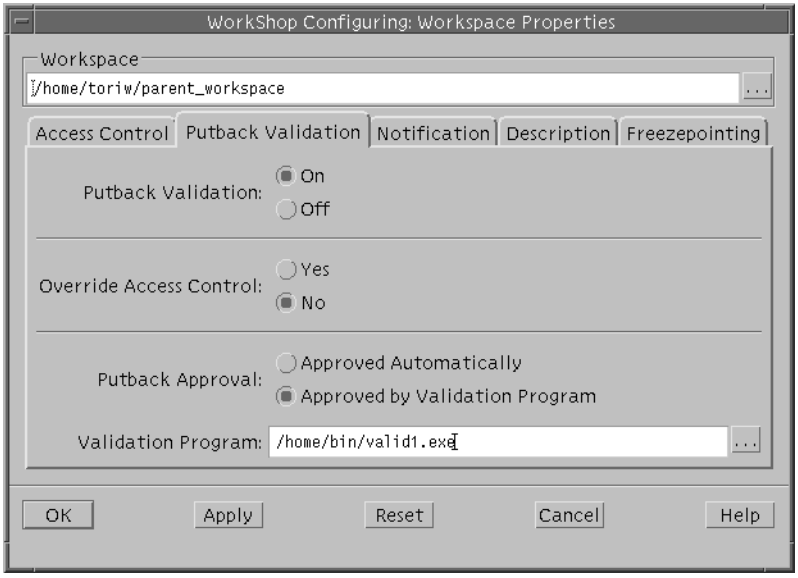


FIGURE 4-2 Workspace Properties: Putback Validation Tab

5. **Click Putback Validation On.**
6. **Click OK.**

Invoking Your Own Putback Validation Program

If a validation program is set for a workspace, any putback operation to the workspace will invoke the program that you indicate. If the program fails, then the putback is canceled.

The validation program is not a part of Sun WorkShop TeamWare. You must provide your own putback validation program. A sample validation program is provided in CODE EXAMPLE 4-1.

To invoke your validation program:

1. **In the Configuring Window, load the workspace by choosing File ➤ Load Workspaces.**
2. **Click on the workspace to select it.**
3. **Choose Workspace ➤ Workspace Properties.**
4. **Click the Putback Validation tab in the Workspace Properties dialog box.**
5. **Click Putback Validation On.**
6. **Click Putback Approval: Approved by Validation Program.**
7. **Type the path to your validation program in the Validation Program text box.**
8. **Click OK.**

Sample Validation Program

This is a sample validation program you can use to determine whether you want to allow a putback to a specific workspace. It can be any executable file (shell script, program, etc.) that accepts arguments passed by putback and returns nonzero exit status to deny putback.

The user who wants to do a putback will be prompted for a Integration Request ID and then the validation program will be invoked with the following arguments:

- User ID of the user doing the putback ($1)
- "from" workspace full path name ($2)
- "to" workspace full path name ($3)
- Integration Request ID ($4)
- Name of the file which contains the list of files to be modified ($5)

CODE EXAMPLE 4-1 is a program that checks all the arguments and, if successful, returns a zero exit status to allow the putback.

CODE EXAMPLE 4-1 Sample Putback Validation Program

```
## Start of sample validation program
#!/bin/sh
##
## Here is the list of users who can perform
## putback to the workspace
```

CODE EXAMPLE 4-1 Sample Putback Validation Program *(Continued)*

```
##
valid_users="nikm azv builder vvg aar"
##
## Here is the list of Integration Request Ids which
## are accepted
##
P1Bugs="1111111 2222222 1234567 bobs_bug"
##
## Here are the directories which cannot change
##
DirsReady="doc subdir/doc"
##
## Save arguments
##
User=$1
        shift
Parent=$1
        shift
Child=$1
        shift
IRI=$1
        shift
Files=‘cat $1‘
##
## Validate user
##
isValid="false"
for u in $valid_users
do
   if [ "$User" = "$u" ] ; then
      isValid="true"
      break
   fi
done
if [ "$isValid" = "false" ] ; then
   # invalid user
   echo ""
   echo "*** Validation failed: User $User is not allowed \
        to putback to $Parent"
   echo ""
   exit 1
fi
##
```

CODE EXAMPLE 4-1 Sample Putback Validation Program *(Continued)*

```
## Validate Integration Request Id (IRI)
##
isValid="false"
for u in $P1Bugs
do
  if [ "$IRI" = "$u" ] ; then
     isValid="true"
     break
  fi
done
if [ "$isValid" = "false" ] ; then
   # invalid IRI
   echo ""
   echo "*** Validation failed: Integration request $IRI \
        is invalid"
   echo ""
   exit 1
fi
##
## Validate files
##
for u in $Files
do
   for uu in $DirsReady
   do
      x=‘echo $u | grep $uu‘
      if [ "$x" != "" ] ; then
         isValid="false"
         echo ""
               echo "*** Validation failed: File $u \
                    cannot be changed"
               echo ""
               exit 1
           fi
       done
done
##
## Exit 0 - putback is allowed
##
exit 0
## End of sample validation program
```

Removing Workspace Locks

To assure consistency, the Configuring transactions—Bringover, Undo, and Putback—lock workspaces during these transactions.

If you attempt to bring over files into a workspace that is locked, you will get a message that states the name of the user who 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)
```

Sun WorkShop TeamWare removes these locks after the transaction is over. Normally, you will not encounter a locked workspace. However, if a transaction encounters problems, sometimes locks stay in place and you can no longer access the workspace. If you are sure no one else is accessing the workspace, but a lock remains, you can remove it manually.

Note – Be sure no one else is modifying the workspace before removing the lock. Removing a lock from a workspace that is currently being modified by another user compromises the integrity of that workspace.

To remove a workspace lock:

1. **Make sure no one else is modifying files or performing any transactions in the workspace.**
2. **In the Configuring Window, load the workspace by choosing File ➤ Load Workspaces.**
3. **Click on the workspace to select it.**
4. **Choose Workspace ➤ Edit Locks.**
5. **In the Edit Locks dialog box, click a lock to select it.**
6. **Click Delete.**
7. **To close the Edit Locks dialog box, click OK.**

CHAPTER 5

Managing Files

Once you have created TeamWare workspaces, you can start work on your project files. Coordinating write access to source files is important when changes will be made by several people. Maintaining a record of file updates allows you to determine when and why changes were made. Versioning is the TeamWare tool you use to manage files. This chapter shows you how to do these basic tasks in Versioning:

- Starting Versioning
- Adding Files to a Workspace
- Checking Out a File
- Editing a File
- Changing Your Default Editor
- Checking In a File
- Reversing Changes to a File
- Integrating Changes By Putting Back Files

Starting Versioning

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

- **To start Versioning, choose TeamWare ➤ Versioning in the Configuring window.**

  When you start Sun WorkShop TeamWare Versioning, the Versioning window opens. The Versioning main window displays the directories and files of the current directory (see FIGURE 5-1).

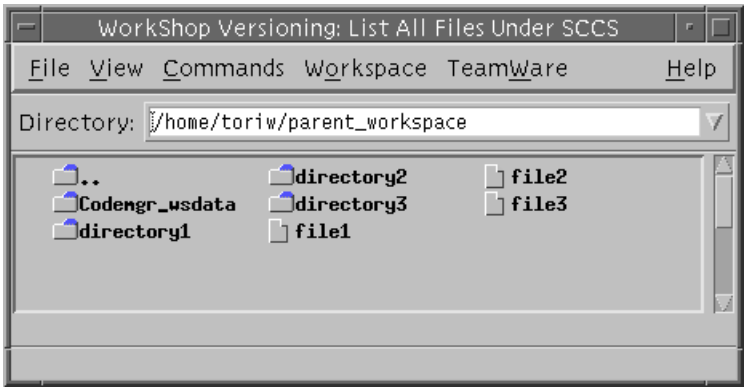


FIGURE 5-1 Versioning Window

The Versioning window displays the directories and files of your project. Versioning displays all directories, but displays only files that are under SCCS control. See "Adding Files to a Workspace" on page 67.

TABLE 5-1 describes the Versioning window menus.

TABLE 5-1 Versioning Window Menus

Menu	Description
File	Provides commands for managing files.
View	Provides commands to control and update the icons shown in the directory pane.
Commands	Provides commands for checking files in and out.
Workspace	Provides commands for finding, deleting, and moving files under SCCS control.
TeamWare	Provides commands for starting other TeamWare too"from" workspace full path name ($2) "to" workspace full path name ($3) Integration Request ID ($4) Name of the file which contains the list of files to be modified ($5)ls.
Help	Provides commands to display online help and release documentation.

Directories are shown whether or not they contain files that are under SCCS control. A directory is a container for files and directories and can possibly contain SCCS files and directories further down the hierarchy. Files are shown only in the Versioning window if they are under SCCS control. To look at the non-SCCS files in a directory, use the Check in New window.

Adding Files to a Workspace

For the Sun WorkShop TeamWare configuration management tools to be able to recognize files, the files must be under SCCS version control.

To add files to a TeamWare workspace:

1. **Start Configuring.**
2. **Load the workspace you want to work with by choosing File ➤ Load Workspaces.**

   If the directory is not yet a TeamWare workspace, see "Creating a Parent Workspace" on page 21.
3. **Click on the workspace to select it.**
4. **Choose TeamWare ➤ Versioning.**
5. **In the Versioning Window, choose Commands ➤ Check In New.**
6. **In the Check In New dialog box, select the files you want to add to the workspace.**

   You can Shift-click to select multiple files.
7. **Click OK.**

   The files you have checked in are now displayed in the Versioning window.

Checking Out a File

Versioning requires that you check out a file before you modify it. When a file is checked out from a workspace, no other user is allowed to check out the file. If you have two users who need access to the same file, have one user create a child workspace so they can have their own copy of the file.

To check out a file:

1. **In the Configuring Window, choose TeamWare ➤ Versioning.**
2. **In the Versioning Window, navigate to the directory where the files exist and select the icon of the first file you want to change.**

   You can shift-click to select several files.

3. **Choose Commands ➤ Check Out.**

   If you choose Commands ➤ Check Out and Edit, Versioning starts up an editor for you.

   A check mark is displayed on the file icon of the selected file. If you want to use a different editor than the default, see "Changing Your Default Editor" on page 68.

Editing a File

To edit a Sun WorkShop TeamWare file:

1. **In the Configuring Window, choose File ➤ Load Workspaces to load your workspace.**
2. **Click the workspace to select it.**
3. **Choose Teamware ➤ Versioning.**

   The Versioning Window appears.
4. **In the Versioning window, select the file you want to edit by clicking on it.**
5. **Choose Commands ➤ Check Out and Edit.**

   This opens the file in the default editor. If you want to use a different editor, see "Changing Your Default Editor."
6. **After you have edited and saved the file, check in the file.**

   See "Checking In a File" on page 69.

Changing Your Default Editor

To change your default editor:

1. **In the Configuring Window, choose TeamWare ➤ Versioning.**
2. **In the Versioning Window, choose View ➤ Options.**
3. **In the Options dialog box, select the editor you want to use.**
4. **If you choose Other, type the path to your editor in the box that appears.**

Checking In a File

To check in a file:

1. **In the Versioning Window, navigate to the directory where the files are stored.**
2. **Click the icon for file you want to check in.**

   You can Shift-click to select multiple files. Files that are checked out have a red check mark on the icon.
3. **Choose Commands ➤ Check In.**
4. **In the Check In dialog box, type a comment.**
5. **Click OK.**

   Sun WorkShop TeamWare saves your comments and checks in the file. To integrate the changes to the parent workspace, see "Integrating Changes By Putting Back Files" on page 70.

Reversing Changes to a File

Versioning also provides a way to return a file to the state it was in before it was checked out. This is useful if you have made changes to a file, but don't want those changes checked back in.

To reverse the changes made to a checked-out file:

1. **In the Configuring Window, choose TeamWare ➤ Versioning.**
2. **In the Versioning Window, navigate to the directory where the checked-out file is stored.**

   The icon of the file should have red check mark, indicating it is checked out.
3. **Click on the file to select it.**

   Shift-click to select multiple files.
4. **Choose Commands ➤ Uncheckout.**

   This reverses the most recent changes to the document and returns the file to its previous state.

Integrating Changes By Putting Back Files

After you have made changes to the files in your workspace, you need to put back these changes to the parent workspace. This makes your changes available to the other members of your team.

To put back files to a parent workspace:

1. **In the Configuring Window, choose Actions ➤ Putback.**
2. **In the Putback tab, select the files you want to put back.**
3. **Click in the Comments text box and type a comment.**

   Depending on how your administrator has set up email notification, many people may see this comment.
4. **Click Putback.**

   The Transactions Output window is displayed and shows information about the putback. If there is a conflict during a putback, Configuring automatically takes you to the Resolve Tab (see Chapter 6).
5. **Click Close to close the Transactions Output window.**

Note – For a more detailed description of performing a putback, see "Putting Back Changes to a Parent Workspace" on page 27.

CHAPTER 6

Resolving Differences Between Files

When two copies of the same file have had changes made to them, these files are in conflict. When you try to put back a file that you have changed in your child workspace, but it has also been changed in the parent workspace, the Sun WorkShop TeamWare tools will prevent you from putting back the file until you have resolved the differences between them. Merging is the tool you use to resolve conflicts between files.

The chapter covers the following topics:

- Starting Merging
- Resolving Conflicts in a Workspace
- Viewing the Merging Window
- Resolving Differences
- Using Automatic Merging
- Undoing Changes
- Merging Options
- Merging Example

Starting Merging

In the Configuring window, a workspace with conflicts has an icon in which one of the files is red and red motion lines indicate a conflict. In FIGURE 6-1, `ws2` has a file in conflict with its parent, `ws1`.

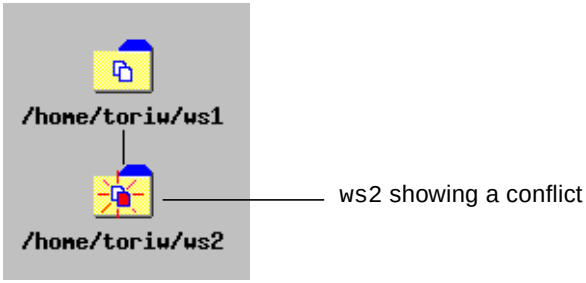


FIGURE 6-1 Workspace Conflict Example

Conflicts need not be resolved immediately. You can continue to make changes and create new deltas in conflicted files in the child workspace. New deltas are created on a branch; when you finally resolve the conflict, the latest delta is the one merged with the version brought over from the parent. *Conflicts must be resolved before you can put back the files to the parent*. If you attempt a putback and files are in conflict, Configuring will prompt you to perform a bringover and then display the conflicted files in the Resolve tab of the Transactions dialog box.

You can start Merging in two places:

- The Teamware menu
- The Resolve tab of the Transactions dialog box

Starting Merging From the TeamWare Menu

To start Merging, Choose TeamWare ➤ Merging in the main window of Configuring, Versioning, or Freezepointing.

Starting Merging From the Resolve Tab

The Resolve tab of the Transactions dialog box acts as intermediary between you and Merging. It lists conflicted files that are detected during Bringover Update transactions. The Resolve transaction allows you to select the conflicted file, and starts Merging with the conflicted files already loaded. Configuring automatically takes you to the Resolve tab of the Transactions dialog box if you chose to resolve a conflict during a putback. You can also open the Resolve tab by choosing Actions ➤ Resolve.

Viewing the Merging Window

When you start Merging, the Merging window appears (see FIGURE 6-2). The Merging window is divided into three panes: two side-by-side panes, which display different versions of the file, and the merged result in the bottom pane. The top two panes are read-only; the bottom pane contains selected lines from either or both versions of the file and can be edited to produce a final merged version.

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

- The child delta is displayed in the left pane labeled Child
- The parent delta is displayed in the right pane labeled Parent

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.

Resolving Conflicts in a Workspace

To resolve a conflict in a workspace:

1. **In Configuring, double-click the icon of a conflicted workspace.**

   The Resolve tab of the Transaction dialog box opens with the names of its conflicted files displayed in the File List pane.

2. **Click Merge Conflicts.**

   Resolve starts Merging and begins to process the list of files in the File List pane. For each file in the list, Configuring extracts the parent delta, the child delta, and the common ancestor from the SCCS history file and passes their path names to Merging. The Merging window opens with the files loaded and ready for merging.

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

   See “Resolving Differences” on page 77 for details about using Merging.

4. **Once you have resolved conflicts, save the file.**

   After you use Merging to resolve differences between the parent and child versions of the file, Configuring creates a new delta in the child SCCS history file. The new delta contains the merged result you created using Merging.

5. **Repeat this process until all conflicts are resolved.**

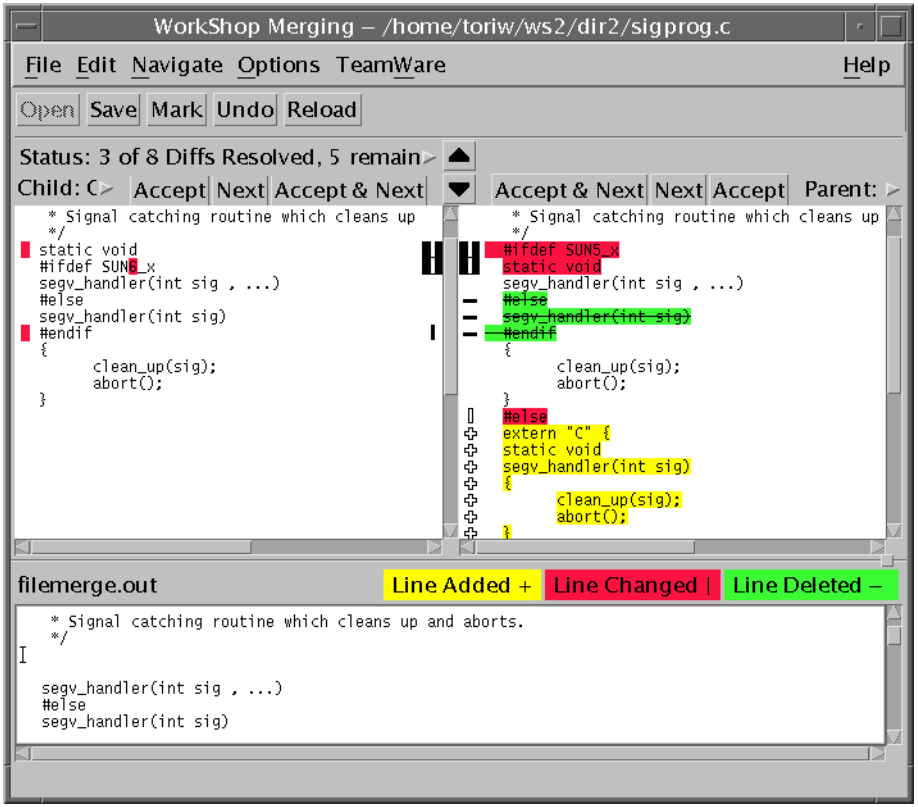


FIGURE 6-2 Merging Window

Reading Merging Glyphs

To help you find differences more easily, Merging highlights lines that differ with color and glyphs. Yellow shows an addition, red shows a change, green shows a deletion.

The meaning of glyphs is different if you are comparing two versions with each other (two input files) or if you identify an ancestor for the two versions of the file (three input files). FIGURE 6-2 shows the Merging window with glyphs indicating differences between two files.

Two Input Files

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

- If two lines are identical, no glyph is displayed.
- If two lines are different, a vertical bar (|) is displayed next to the line in each input text pane, and the different characters are highlighted in yellow.
- If a line appears in one file but not in the other, a plus sign (+) is displayed next to the line in the file where it appears, and the different characters are highlighted in red.
- Resolved differences are marked by glyphs in outline font.

Three Input Files

When you load two files to be merged, you can also specify a third file, called the ancestor of the two files. An ancestor file is any earlier version of the two files. When you identify an ancestor file, it is used as a basis to compare the two files and automatic merging can be done. Merging marks all lines in the derived files or their descendants that differ from the ancestor and produces a merged file based on all three files.

The lines in the files that are different from the ancestor file are marked with change bars and colors. Here's what each means:

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

Loading Files Into Merging

If files are not automatically loaded into Merging by Resolve, you can load files by choosing File ➤ Open or clicking the Open button. The Open Files dialog box is displayed (see FIGURE 6-3).

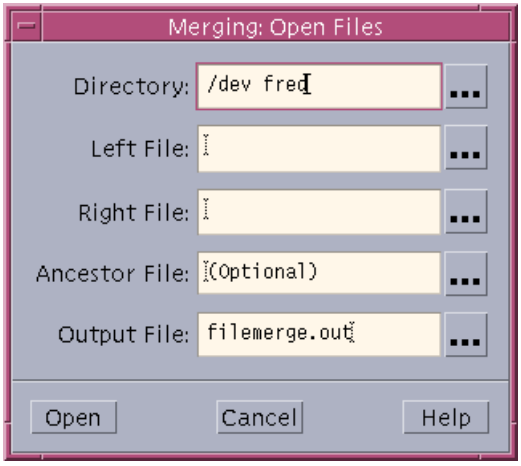


FIGURE 6-3 Merging: Open Files Dialog Box

TABLE 6-1 describes the text boxes in the Open Files dialog box.

TABLE 6-1 Merging Open Files Dialog Box Text Boxes

Text Box	Description
Directory	Shows the current working directory whenever you start Merging from Sun WorkShop or from the command line with no arguments. You can edit this field. Merging interprets the file names you specify in the window as relative to the current working directory. Therefore, you can use constructs such as *subdir/filename* to specify a file in a subdirectory and *../filename* to specify a file in a parent directory. Any file name you specify that begins with a "/" character is interpreted as an absolute path name, not as relative to the current working directory.
Left File	Lets you specify the file to appear in the left text pane, also considered the child pane.

TABLE 6-1 Merging Open Files Dialog Box Text Boxes *(Continued)*

Text Box	Description
Right File	Lets you specify the file to appear in the right text pane, also considered the parent pane.
Ancestor File	Lets you specify the name of an ancestor file. If you type a file name in this text box, Merging compares the file to the files to be merged and identifies lines in those files that differ from the ancestor. The automerged file is based on the ancestor file, but the ancestor file itself is not displayed in any Merging window. If you do not type an ancestor file name, Merging compares only the left and right files and derives the output file from them. Automerging is not possible without an ancestor file.
Output File	Lets you specify the name for the merged output file. Merging uses the name `filemerge.out` unless you specify a different name, and stores the file in the current working directory.

In a loaded Merging window, the names of the left file, right file, and output file are displayed above the appropriate text panes. The name of the ancestor file (for a three-way diff only) is displayed in the window header.

Resolving Differences

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

A difference is represented by a blank line in the merged (output) file in the lower text pane. To resolve a difference, you edit the line displayed by either:

- Accepting the line displayed and incorporating it into the merged file by clicking either the Accept or Accept & Next button over the pane you want to accept.
- Editing the line in the merged file by hand, and marking the difference as resolved by choosing Edit ➤ Mark Selected as Resolved.

If you want Merging to automatically put lines that are not in conflict (that is, the line has changed in one file but not in the other) into the merged file, you can select Options ➤ Auto Merge. Then when you start Merging, all the resolved lines are put in the merged file for you. For more information, see "Using Automatic Merging" on page 79.

To resolve differences between files:

1. **Determine which difference you are dealing with:**
   - Read the Status Line on the upper left side of the Merging window.
   - See which glyph is highlighted.

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

2. **Choose a version and accept it:**
   - To accept the child version of the line, click the Accept button above the file on the left.
   - To accept the parent version of the conflicted line, click on the Accept button above the file on the right.

   The version you accept will appear in the merged file in the bottom pane.

3. **Click Next to go to the next conflict in the file.**

   To move between differences:
   - Click one of the Next buttons.
   - Use the Navigate menu.
   - Use the Diff Navigator (the up and down arrows above where the parent and child panes meet).

   If you do not want your changes and you want to start over, click the Reload button. This ignores all the conflicts that you have resolved and reloads the files.

4. **Click Save when you have resolved all conflicts.**

Using Automatic Merging

If you have loaded a common ancestor file, Merging is often able to resolve differences automatically, based on the following rules:

- If a line has not been changed in either descendant (it is identical in all three files), it is placed in the merged file.
- If a line has been changed in only one of the descendants, the changed line is placed in the merged file. A change could be the addition or removal of an entire line, or an alteration to some part of a line.
- If identical changes have been made to a line in both descendants, the changed line is placed in the merged file.
- If a line has been changed differently in both descendant files so that it is different in all three files, Merging places no line in the merged file. You must resolve the difference—either by using a line from the right or left file, or by editing the merged file by hand.
- Resolved differences are marked by glyphs in outline font.

Automerging Rules Summary

TABLE 6-2 summarizes the automerging algorithm.

- Ancestor is the version of a text line that is in the ancestor file
- Change 1 is a change to that line in one of the descendants
- Change 2 is another change, different from Change 1.
- When a line is changed differently in the left and right descendants, automerging does not put either line in the merged file.

TABLE 6-2 Automerging Rules Summary

Left Descendant	Right Descendant	Automerged Line
Ancestor	Ancestor	Ancestor
Change 1	Ancestor	Change 1
Ancestor	Change 2	Change 2
Change 1	Change 1	Change 1
Change 1	Change 2	No Automerge

Undoing Changes

You can undo changes with the Edit ➤ Undo command.

You can also use the Reload button on the Merging window tool bar to ignore all edits that have been performed on the two files and reload them from disk. Any nonconflicting differences will be displayed in the bottom pane if the Auto Merge option is selected.

Merging Options

You can change the behavior of Resolve and Merging in two places:

- Resolve Options in Configuring
- Display Options in Merging

Resolve Options

The Resolve tab of the Tool Properties window (FIGURE 6-4) lets you change the behavior of the Resolve pane of the Transactions window. The specific properties are described in TABLE 6-3.

FIGURE 6-4 Tool Properties Dialog Box: Resolve Tab

TABLE 6-3 Tool Properties Dialog Box: Resolve Tab

Property	Description
Auto Start Merging Window	Causes Merging to start automatically when you select the Resolve transaction pane.
Auto Advance	Causes the next file in the list to be automatically loaded into Merging after the current file is resolved.
Prompt for Checkin Comments	A default comment is automatically supplied during check-in after you resolve a file. This property causes you to be prompted for an additional comment that is appended to the standard comment.
Use Existing Merging Window	If this property is set, a running Merging process is reused during subsequent resolve operations.
Auto Save (when no unresolved diffs)	If this property is set, and all the changes in the file can be "automerged," the files will also be saved and checked in.

Display Options in Merging

The Options menu in the Merging window contains the following components. The first five options on this menu are toggles, that is, you can turn them on and off by selecting them. A small box appears to the left of an option when it is on.

TABLE 6-4 lists the Merging Display options.

TABLE 6-4 Merging: Display Options

Property	Description	
Auto Merge	Automatically resolves any nonconflicting differences and constructs a merged version in the Merged Result Pane. The default is on.	
Scroll Together	Lets you set the text panes so they scroll together (corresponding lines are always aligned in each window) or scroll separately. The default setting is Scroll Together.	
Show Line Numbers	Displays line numbers in the unmerged files.	
Show Line Ends	Displays a small black box at the end of each line in the unmerged files.	
Show Diff Navigator	Displays the Diff Navigator between the two unmerged files. The Diff Navigator displays differences between the two files as colored lines. Click the slide boxes on either side of the Diff Navigator to scroll through either file, or click the arrows on the top or bottom to move the same distance in both files.	
Tab Display	Lets you customize tab stops. You can choose:	
	Control Character (^I)	Displays the ^I control character for each tab space. The default setting is on.
	Spaces	Allows you to set the number of spaces in each tab stop to 1, 2, 3, 4, 6, 8, 10, 12, or 16. The default is 8.
Diff Options	Lets you customize diff behavior. You can choose:	
	Ignore trailing white space	Ignores trailing white space when finding lines that differ. The default setting is off.
	Ignore all white space	Ignores embedded and trailing space when finding lines that differ. The default is off.
	Suppress case sensitivity	Ignore letter case when finding lines that differ. The default is off.

Merging Example

This example merges two files that have a common ancestor. The files are `file_1` and `file_2`, and the ancestor file is named `matriarch`. The descendant files `file_1` and `file_2` were derived from `matriarch` by editing. The edits show all varieties of changes that could occur in the descendants: deleting lines, adding new lines, and changing lines.

The content of each line in the example helps to identify whether or not it was changed, and how. The ancestor file contains only twelve lines and is shown in CODE EXAMPLE 6-1.

Merging does not number lines in the files it loads; the numbers are part of the example text and were placed there for clarity.

CODE EXAMPLE 6-1 Ancestor File (`matriarch`)

```
 1 This line is deleted in file_1
 2 This line is in all three files
 3 This line is deleted in file_2
 4 This line is in all three files
 5 This line is in all three files
 6 This line is changed in descendants
 7 This line is in all three files
 8 This line is changed in descendants
 9 This line is in all three files
 10 This line is changed in file_2
 11 This line is in all three files
 12 This line is in all three files
```

CODE EXAMPLE 6-2 shows the contents of `file_1`. This file is identical to `matriarch` with the following exceptions:

- The line numbered 1 in the `matriarch` file was deleted in `file_1`.
- A new line was added following the line numbered 4.
- The line numbered 6 was changed (a different change was made to this line in `file_2`).

- The line numbered 8 in the `matriarch` file was changed (an identical change was made to this line in `file_2`).

CODE EXAMPLE 6-2 Descendant File (`file_1`)

```
 2 This line is in all three files
 3 This line is deleted in file_2
 4 This line is in all three files
  &&& Added to file_1 &&&
 6 This line is modified in file_1 from matriarch
 5 This line is in all three files
 7 This line is in all three files
 8 #&# Changed in file_1 and file_2 #&#
 9 This line is in all three files
 10 This line is changed in file_2
 11 This line is in all three files
 12 This line is in all three files
```

CODE EXAMPLE 6-3 shows the contents of `file_2`. This file is identical to `matriarch` with the following exceptions:

- The line numbered 3 in the `matriarch` file was deleted.
- The line numbered 6 was changed (a different change was made to this line in `file_1`).
- The line numbered 8 was changed (an identical change was made to this line in `file_1`).
- The line numbered 10 was changed (no change was made to this line in `file_1`).
- A new line was added following the line numbered 11.

CODE EXAMPLE 6-3 Descendant File (`file_2`)

```
 1 This line is deleted in file_1
 2 This line is in all three files
 4 This line is in all three files
 5 This line is in all three files
 6 This line is altered in file_2 from matriarch
 7 This line is in all three files
 8 #&# Changed in file_1 and file_2 #&#
 9 This line is in all three files
 10 ### Changed in file_2 ###
 11 This line is in all three files
  ### Added to file_2 ###
 12 This line is in all three files
```

In the upper left of the Merging window, Merging has reported finding seven differences, of which only one remains unresolved (see FIGURE 6-5). Six differences were resolved by automerging and are marked by glyphs in outline font (see FIGURE 6-6 and FIGURE 6-7).

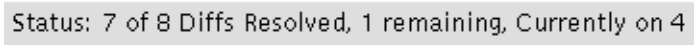


FIGURE 6-5 Merging Status of `file_1` and `file_2` After Automerging

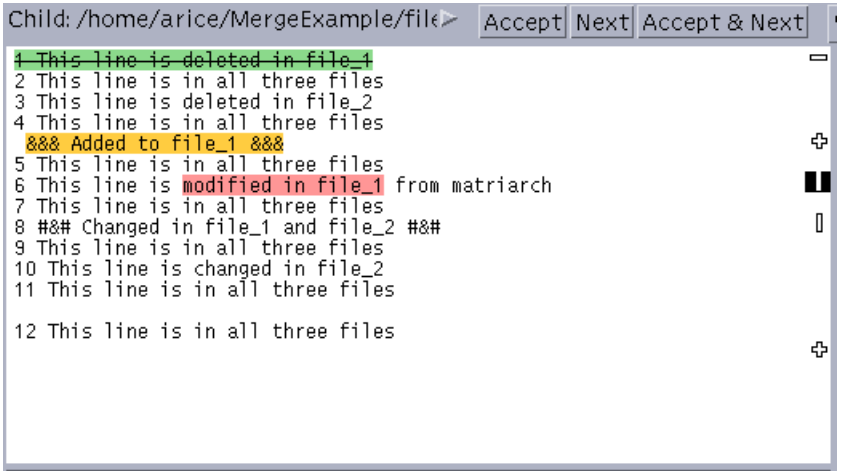


FIGURE 6-6 `File_1` Displayed in Child Pane After Automerging

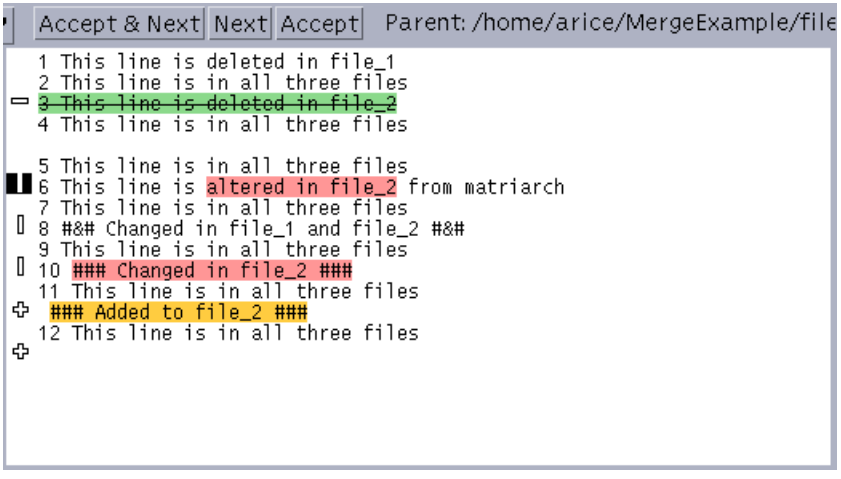


FIGURE 6-7 `File_2` Displayed in Parent Pane After Automerging

The meaning of the glyphs is as follows: a vertical bar means a change in the marked line, a plus sign signifies a line added, a minus sign means a line was deleted. Unresolved states are marked by solid glyphs, unresolved by outline. These glyphs are highlighted in color except when the color map is full. The default significance is: red indicates a change, green indicates a deletion, yellow shows an addition.

The unresolved difference (line 6) is marked by a vertical bar.

Examining Differences

Merging highlights the unresolved difference, which it identifies as the line numbered 6 in `file_1` and `file_2`. When differences are being resolved with Merging, the resulting Merging window (`filemerge.out`) shows the current state of the file with automatic merging.

You can proceed to the next unresolved difference by clicking the down arrow above the appropriate file or choosing Navigate ➤ Next ➤ Difference. The next difference becomes the current difference.

You can proceed through the differences by clicking on the down arrow.

Automerging preserves a change that was made to one file if no change was made in the other file. When a difference has not been resolved by automerging, as indicated by the solid highlighted glyph next to the lines involved in the difference, you need to resolve the difference by making a choice. The vertical line indicates that the line has been changed (as opposed to added or deleted). Automerging does not include either line in the merged file because the same line was changed differently in the two files. You will have to determine which change to accept.

Resolving a Difference

You could resolve this difference in one of the following ways:

- Clicking the Accept or Accept & Next button on the left to place the line from `file_1` into the output file
- Clicking the Accept or Accept & Next button on the right to place the line from `file_2` into the output file
- Editing the output file by hand

Editing the Output File

To edit the output file:

1. **Move the pointer into the output file's text pane and place it in the line you want to change.**

   In this example, the following line was typed in:

   ```
   >>> This line edited by hand <<<
   ```

2. **Choose Edit ➤ Mark Selected as Resolved.**

   This menu item marks the difference as resolved. In this example there are no more unresolved differences, so the next difference remains the current one.

   The message in the upper left part of the window now indicates that all differences have been resolved.

3. **Verify the automerged differences.**

   Navigate through the differences by clicking the down arrow.

   The final difference results from a line that was added only to `file_2`. Merging would place the new line in the output file, just as it did when a new line was added to `file_1`, which resulted in the third difference.

CHAPTER 7

Advanced File Management

This chapter shows you how to perform advanced file management, including:

- Updating the Files in Your Workspace
- Viewing File History
- Renaming, Moving, or Deleting Files
- Creating a Customized Menu
- Adding a Path to the Load Menu
- Changing Versioning Properties
- Setting SCCS File Properties

Updating the Files in Your Workspace

Once you have created a workspace, you need to update it regularly using the Bringover transaction.

To update your workspace:

1. **In the Configuring Window, choose Actions ➤ Bringover Update.**
2. **Click the Bringover Update tab in the Transactions dialog box.**
3. **Confirm that the parent and child workspace directories are correct.**
4. **Select the directories and files to update.**
5. **Click Bringover.**

   For a detailed explanation of how to perform this transaction, see “Updating a Child Workspace (Bringover Update)” on page 25.

Viewing File History

Versioning allows you to visually peruse the history of a file in a Sun WorkShop TeamWare workspace. This way, you can see who made changes to a file, when the changes were made, and (with good comments) why changes were made.

The History window (see FIGURE 7-1) displays an illustration of 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.

Using the History window, you can:

- Select a delta from the history graph that will display information about the delta in the Delta Details 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 View ➤ Show File Contents. This opens an editor window with the contents of the selected delta displayed.
- Select two deltas and choose View ➤ Diff in Merging Window. This displays a Merging window in which the two selected deltas are displayed side by side for comparison.
- Select two deltas and choose View ➤ Diff in Text Window. An editor window opens up displaying the textual differences from the SCCS `diffs` command.
- Select two deltas and choose View ➤ Diff with Context. An editor window opens up displaying the textual differences from the SCCS `diffs -c` command.

To view the history of a file:

1. **In the Versioning Window, click a file to select it.**
2. **Choose File ➤ File History.**

   This displays the File History Window (see FIGURE 7-1).
3. **When you want to view another delta of the same file, click a delta (represented by a number, for example, 1.2) in the History Graph pane.**

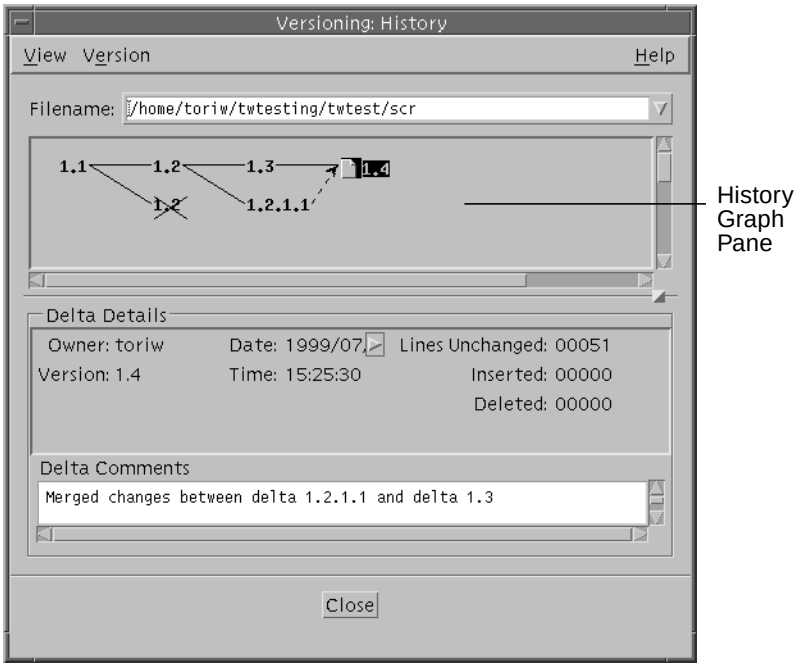


FIGURE 7-1 File History Window

File History Window

TABLE 7-1 describes the items in the File History window:

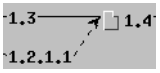
TABLE 7-1 File History Window

Item	Description
View menu	Provides commands for managing files.
Version menu	Provides commands for managing icons in the History Graph pane.
Filename text box	Displays the file path name.
History Graph pane	Displays icons for file deltas.
Delta Details pane	Contains delta history information.
Delta Details pane	Contains delta history information.

File History Viewer Symbols

TABLE 7-2 lists the symbols you can see in the File History Viewer window.

TABLE 7-2 File History Viewer Symbols

Symbol	Description
	A file icon to the left of the version number indicates the default delta.
	A red check mark indicates the file is checked out.
	Crossed out deltas are unmerged.
	Solid lines represent the default deltas path.
	Dotted lines with an arrow represent merged branches.
	Double underline indicates the default branch (set by the `sccs-admin` flag `-d`). See "Setting SCCS File Properties" on page 102.

How To Read a File's History: Deltas, Branches and Versions

The Versioning tool keeps track of the various versions of a file for the entire life of the file. Each time you check a file in, Versioning records the line-by-line differences between the file you check in and the previous version of the file. This line-by-line difference is known as a *delta*.

When you check a file in to Sun WorkShop TeamWare for the first time, it is numbered 1.1 (by default). Successive deltas are numbered 1.2, 1.3, and so forth. The straight progression from 1.1, 1.2, 1.3, etc. is know as the trunk of an SCCS tree. There are times, however, when it is necessary to create an alternative branch off of the trunk. *Branches* are used to allow you to develop two different versions of the same file in parallel, often for bug fixes in source code. Branches are numbered from where they split off from the tree, for example, 1.2.1.1.

An SCCS delta ID (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 increased automatically. 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, 1.2.1.1 identifies the first delta of the first branch derived from delta 1.2. A second branch to this delta would be numbered 1.2.2.1 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 1.3.1.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).

The terms delta and version are often used synonymously; however, their meanings are not the same. Versioning constructs a *version* of a file from a set of accumulated deltas. It is possible to retrieve a version of a file that omits certain deltas.

Merging Deltas

There are times when it makes sense to continue development in parallel, and there are times when it makes sense to merge changes back into the main trunk. Once you have merged the files, this is known as a branch closure.

To merge two deltas:

1. **Start Versioning and select the working directory.**
2. **Choose Commands ➤ Check Out to check out a file.**
3. **Choose File ➤ File History to display the history graph of the file.**
4. **Select two deltas from the graph in the History window.**
5. **To inspect the differences, choose one of the following:**
   - View ➤ Diff in Merging Window
   - View ➤ Diff in Text Window
   - View ➤ Diff with Contents
6. **Make changes to the file.**
7. **Add necessary comments.**
8. **Choose Commands ➤ Check in to check in the file.**

Renaming, Moving, or Deleting Files

When you rename, move, or delete files, Configuring tracks those changes and manages the altered files during Bringover and Putback transactions. Although Configuring processes these files automatically, it is helpful for you to understand some of the ramifications of renaming, moving, or deleting files.

Note – The best way to delete and rename files is to use the Move and Delete commands from the TeamWare Versioning menu, rather than with the Common Desktop Environment (CDE) FileManager or the SunOS operating system commands `mv` or `rm`. Using the procedures detailed below will maintain the correct relationships between files and accurate file histories.

Renaming or Moving Files

When you bring over or put back files that you (or another user) have renamed or moved, Configuring must determine whether the files have been newly created or whether they existed previously and have been renamed or moved. When you rename or move a file, Configuring updates both the file name and history file for that file. Configuring propagates the name change throughout the workspace hierarchy using the same rules used with file content updates and conflicts.

During transactions, Configuring processes files individually. When you rename or move a directory, each file in the directory is evaluated separately as if each had been renamed or moved individually.

Example

In FIGURE 7-2, the name of file C in the parent is changed to D. When Configuring brings the file over to the child, it must determine 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.

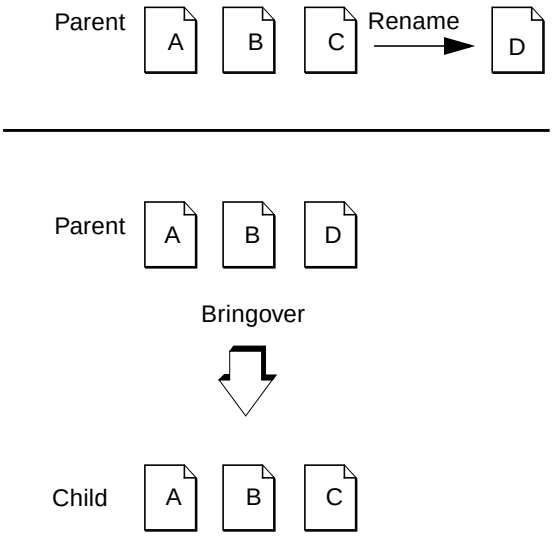


FIGURE 7-2 File “C” Renamed to “D”

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

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

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

```
Examined files:
```

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

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

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

Name History

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

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

Rename Conflicts

In rare cases, a file's name is changed concurrently in parent and child workspaces. This is referred to as a *rename conflict*. For example in FIGURE 7-3, the name of file "C" is changed to "D" in the parent, and concurrently to "E" in the child.

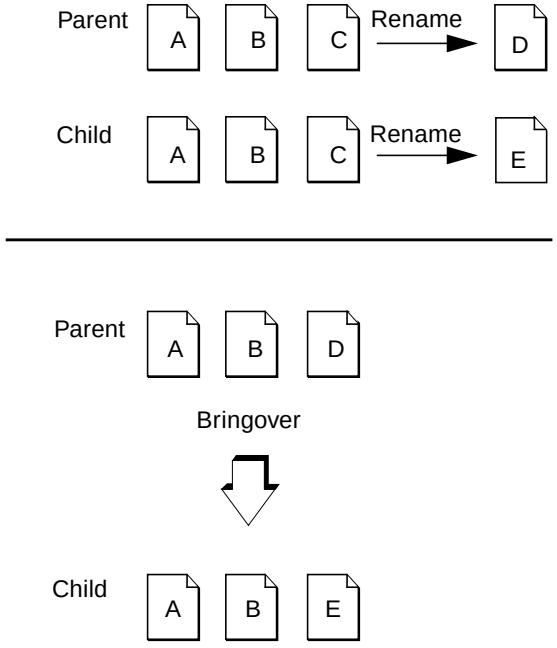


FIGURE 7-3 File "C" is Concurrently Renamed in both Parent and Child Workspaces.

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

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

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

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

Deleting Files

Deleting files from a Configuring workspace is a little trickier than it first appears. Deleting a file from a workspace with something other than Sun WorkShop TeamWare commands causes Configuring to determine that the file has been newly created in the workspace’s parent or child.

In the example in FIGURE 7-4, the file “C” is removed from the child workspace using the SunOS operating system command `rm`; later the Bringover Update transaction is used to update the child.

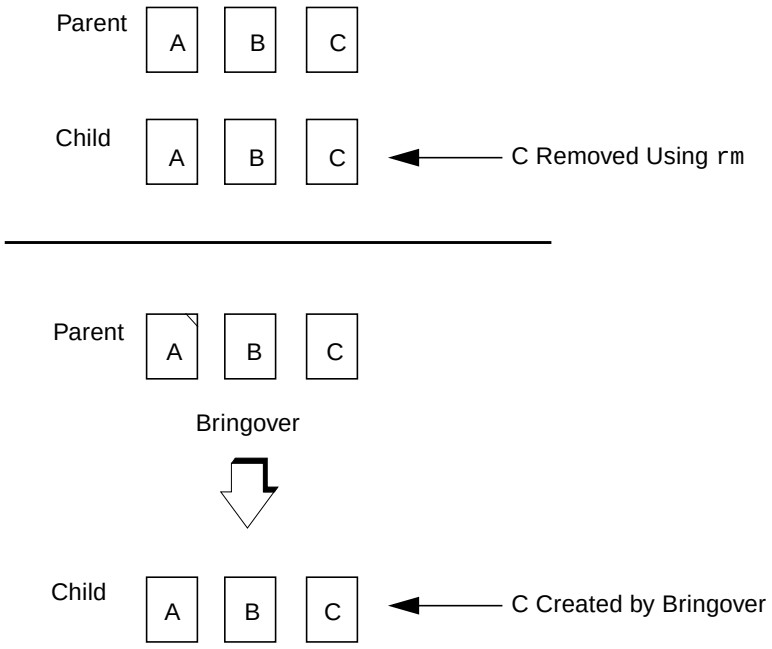


FIGURE 7-4 File "C" is Removed From the Child Using the `rm` Command, Then Recreated by Bringover.

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

Always use Sun WorkShop TeamWare commands to delete files and workspaces.

Deleting a Sun WorkShop TeamWare File

Use this procedure to delete files, rather than using operating system commands. This procedure will maintain the relationships and history of the files.

To delete a Sun WorkShop TeamWare file:

1. **Start Versioning.**
2. **Click on a file.**
3. **Choose Workspaces ➤ Delete Files.**
4. **Click OK.**

Rather than actually deleting the file, Versioning moves the file to a `deleted_files` directory. This change gets propagated throughout the workspace hierarchy as a rename, "deleting" the file in all workspaces.

Using this procedure, you don't have to worry about a file reappearing after you have deleted it (see "Deleting Files" on page 97).

Creating a Customized Menu

In Versioning, you can create your own pull-down menu to access other programs or frequently used commands.

To create a customized menu:

1. **In the Configuring window, choose TeamWare ➤ Versioning to open the Versioning window.**
2. **In the Versioning window, choose View ➤ Options.**
3. **Click the Customized Menu tab.**
4. **In the Customized Menu tab, type the full path name of the command. For example:**

   ```
   /bin/grep
   ```

   Sun WorkShop TeamWare provides two macros to use as arguments to the command: `ARG` and `FILE`.

   For example:

   ```
   /bin/grep $ARGS $FILES
   ```

5. **Type the name you want to appear in the menu in the Menu Label box.**
6. **Check the Output Window check box if you want to see the output from your command.**
7. **Check the Prompt Window check box if you want to prompt the user for input.**
8. **Type the text you want to appear in the prompt window in the Prompt box.**
9. **Click Add to add the menu item to the list.**
10. **Click Test to test your command.**
11. **Click OK.**

    A new menu titled "Customized" appears in the Versioning window. The command that you have added is on this menu.

Adding a Path to the Load Menu

Some projects have a complex structure, and it becomes burdensome to click down several directories or type a long path to get to the files you regularly work with. Instead, you can add a directory to the File ➤ Load menu.

To add a directory to the Load menu:

1. **In the Configuring window, choose TeamWare ➤ Versioning to open the Versioning window.**
2. **In the Versioning window, choose View ➤ Options and click the Load Menu Defaults tab.**
3. **In the Pathname text box, type the full path name of the directory.**

   For example:

   ```
   /set/pubs/Work/Workspaces/TAZ/IntA/sig_team
   ```

4. **In the Menu Label text box, type the name you want to appear on the Load menu.**

   For example:

   ```
   sigteam
   ```

5. **Click Add.**

   The menu label appears in the Load Menu Defaults tab.

6. **Click OK.**

   Your menu label is now on the File ➤ Load menu. Select this menu label and Versioning loads the directory.

Changing Versioning Properties

You can use the Options dialog box (see FIGURE 7-5) to set the Versioning properties. To open the Options dialog box, choose View ➤ Options.

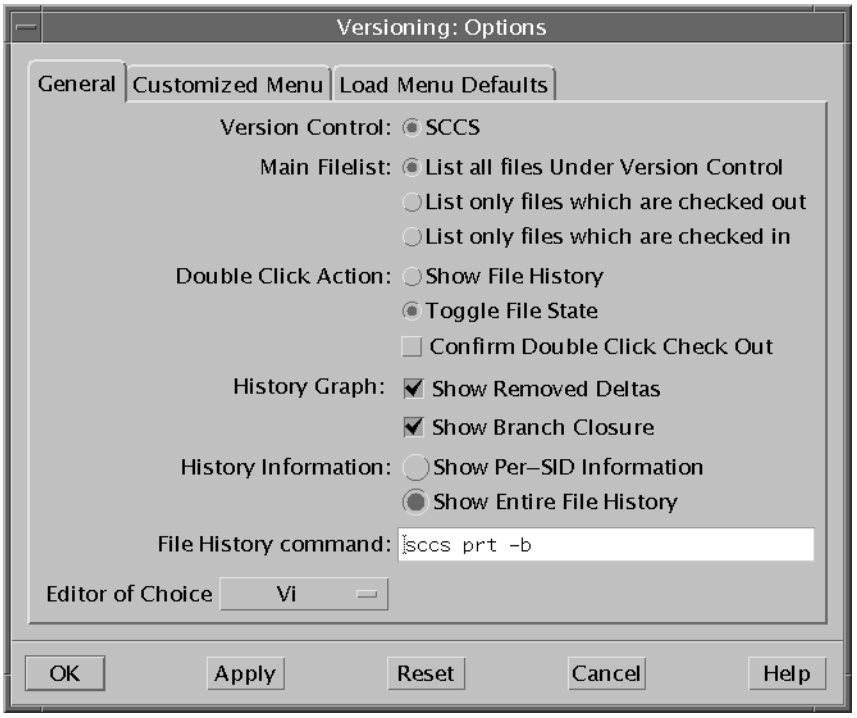


FIGURE 7-5 Versioning Options Dialog Box

TABLE 7-3 describes the items in the General tab in the Versioning Options dialog box.

TABLE 7-3 Versioning Options Dialog Box: General Tab

Item	Description
Version Control button	Currently accepts only SCCS.
Main Filelist buttons	Lets you specify the type of SCCS files displayed in the Versioning window.
Double Click Action buttons	If you select Toggle File State, you can also click Confirm Double Click Check Out check box if you want the check mark to be displayed on the file icon until you click OK in the dialog box.
History Graph check boxes	Lets you define items for display on the history graph.

TABLE 7-3 Versioning Options Dialog Box: General Tab *(Continued)*

Item	Description
History Information buttons	Lets you select type of history information you want displayed. If you select Show Entire File History, you can also specify a command to gather the history.
File History command text box	Available when you select the Show Entire History File History button. Lets you specify the options to the `sccs prt` command used to display the history file. The default is `sccs prt` with no options.
Editor of Choice menu	Lets you specify an editor that automatically starts up when you view the contents of a delta or open a delta to edit. If you select Other, you must type a command in the text box that will start up your editor in a separate window. The file name is appended to the supplied command.

Setting SCCS File Properties

Use the SCCS File Properties dialog box to change SCCS flags. These are options to the `SCCS-admin` command. You can learn more about SCCS file properties by reading the *Solaris Programming Utilities Guide* or with the command `man sccs-admin`.

To set SCCS File Properties:

1. **In the Versioning Window, select a single file.**
2. **Choose File ➤ File Info to open the SCCS File Properties dialog box.**
3. **Set any of the properties in the SCCS File Properties dialog box.**

TABLE 7-4 describes the items in the File Properties dialog box.

TABLE 7-4 SCCS File Properties

Flag	Description
Force Encoding ('e' flag)	This field is read only. Values are Yes and No.
Treatment of 'No id keywords' ('i' flag)	Tells SCCS to issue either an error or a warning when it encounters a file with no id keywords.
Empty Releases ('n' flag)	Creates empty releases when releases are skipped. Values are Allowed and Not Allowed.

TABLE 7-4 SCCS File Properties *(Continued)*

Flag	Description
Branch Deltas ('b' flag)	Enables branch deltas. Values are Enabled and Disabled.
Concurrent Updates ('j' flag)	Allows concurrent updates. Values are Allowed and Not Allowed.
Ceiling on the Releases ('c' flag)	Sets a ceiling on the number of releases that can be checked out. Type a number from 1 to 9999. The default is 9999.
Floor on the Releases ('f' flag)	Sets a floor on the number of releases that can be checked out. Type a number from 1 to 9999. The default is 1.
Default SID ('d' flag)	Sets a default delta number, or SID. For example, you could enter 1.6. For a description of SIDs, see "How To Read a File's History: Deltas, Branches and Versions" on page 92.
Lock Releases ('l' flag)	Locks the release against deltas. Any attempt to check out and edit the file will fail. Type a release number or list of release numbers separated by commas. (For example: 2.1, 2.1.3).
'Q' Keyword Value ('q' flag)	Sets a value for a keyword when the file is opened read-only.
Module Name ('m' flag)	Sets a value for the module name keyword. The default is the SCCS file name with the leading s removed.
Module Type ('t' flag)	Sets a value for the module type.
Validation Program ('v' flag)	Sets a validation program for the MR (Modification Request) numbers associated with a new delta. When you attempt to check in the file, Versioning will prompt you for an MR for the file and pass the MR to the validation program. If the validation program is successful, then the check in is allowed.

CHAPTER 8

Using Freezepointing

With Freezepointing, you can preserve a “snapshot” of the files in your workspace and then retrieve that version of the file at a later time.

The chapter contains the following sections:

Introduction to Freezepointing

During the software development process, it is often useful to create freezepoints of your work at certain times in the release cycle. Freezepoints serve as snapshots of a project that enable you to later re-create 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 files using the standard backup utilities. This method is effective, but it requires a large amount of storage resources and time. With Freezepointing, you preserve freezepoints quickly and simply, using a small amount of storage resource.

This chapter covers the Freezepoint tool, which has a graphical user interface (GUI). You can also issue freezepoint commands from the command line (see Chapter 11).

How Freezepointing Works

Freezepointing lets you create freezepoint files from workspaces. At a later time you can use the freezepoint files to re-create the files and directory hierarchies contained in the workspaces. By default, Freezepointing extracts only files and directories; that is, it retrieves the delta of each file without the file history. Freezepointing gives you the option of recreating a workspace containing the histories for all extracted files.

Creation Defined

When you create a freezepoint file, you specify directories and files to include in the Directories and Files pane of the Freezepointing window (see FIGURE 8-2). Freezepointing recursively descends the directory hierarchies and identifies the most recently checked-in deltas in each history file. Freezepointing then creates a freezepoint file that consists of a list of those files and unique numerical identifiers for each delta (see "Reading Freezepoint Files Format" on page 116).

Freezepointing saves the most recently checked-in delta of a file. This may or may not be the same as the default delta. In the example below, the default delta is 1.3. If 1.2.1.1 is the last delta that has been checked in, Freezepointing will save 1.2.1.1.

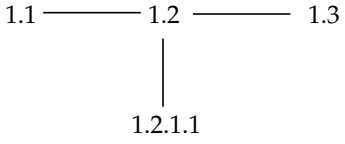


FIGURE 8-1 Which Delta Freezepointing Saves

Extraction Defined

After you have created a freezepoint file, you can use it to retrieve your files from it. You specify the name of the freezepoint 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.

Note – Because a freezepoint file is only a list of differences, you must have the original workspace (or its parent or child) to extract from a freezepoint file.

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 files defined in the original history files; the history files themselves are not recreated unless you ask Freezepointing to create a workspace while performing the extraction. Deltas are extracted from 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 files are later extracted to recreate the hierarchy. You can specify an alternate source directory at the time you perform the extract operation.

Destination Directory

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

Starting Freezepointing

You can start Freezepointing by:

- Typing `twfreeze` at a shell command prompt, followed by the ampersand symbol (&)
- Choosing TeamWare ➤ Freezepointing in the Configuring, Merging, or Versioning window

The Freezepointing window opens with the Creation tab displayed (see FIGURE 8-2).

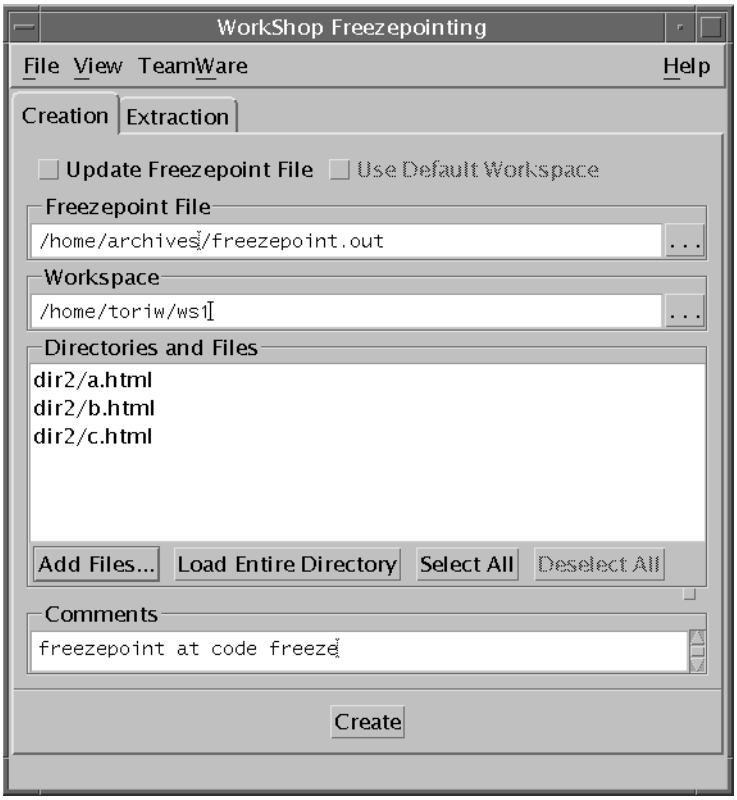


FIGURE 8-2 Freezepointing Window: Creation Tab

TABLE 8-1 lists the items in the Freezepointing Creation tab.

TABLE 8-1 Freezepointing Creation Tab

Item	Description
File menu	Provides a command to exit Freezepointing.
View menu	Provides the Show Output command, where you can view and save status and error messages.
TeamWare menu	Provides commands for starting other TeamWare tools.
Help menu	Provides commands to display help.
Update Freezepoint File check box	Lets you choose to update an existing freezepoint file rather than create a new one.
Use Default Workspace check box	Lets you use the workspace named in the freezepoint file.

TABLE 8-1 Freezepointing Creation Tab *(Continued)*

Item	Description
Freezepoint File text box	Lets you type the absolute path name of the freezepoint file.
Workspace text box	Lets you specify the source workspace.
Directories and Files pane	Contains a list of files and directories that will be preserved in the freezepoint file.
Add Files button	Opens the Add Files dialog box, where you can select files to add to the Directories and Files pane.
Load Entire Directory button	Loads the entire workspace directory.
Select All buttons	Selects all the files and directories listed in the pane.
Deselect All button	Deselects all the files and directories listed in the pane.
Comments text box	Lets you include a comment with the freezepoint file.
Create button	Creates the freezepoint file. Displays the Freezepoint Output box with messages about the creation.

Creating a Freezepoint File

To create a Freezepoint file:

1. **In the Configuring Window, choose TeamWare ➤ Freezepointing to open the Freezepointing window.**

   The Creation tab is displayed (see FIGURE 8-2).

2. **You can accept the default name of `freezepoint.out` or type the name of the freezepoint file in the Freezepoint File box.**

   When you start Freezepointing, the Freezepoint File text box is automatically set to contain the file `freezepoint.out` appended to the path name of the directory from which Freezepointing was started. You can enter your own path or file name. Path names that are not absolute are assumed to be relative to the directory in which Freezepointing is started.

3. **Type the name of the source workspace in the Workspace text box.**

   When you start Freezepointing, the Workspace text box is automatically set to the workspace you have specified using the `CODEMGR_WS` environment variable. If the variable is not set, and the directory from which Freezepointing is started is hierarchically within a workspace, the Workspace text box is initialized with the path name of that workspace.

4. **Create a list of the directories and files that you want to preserve in the Directories and Files pane.**

   Click the Add Files button to open the Add Files dialog box.

   a. **Select the files you want to include.**

      Click a file name to select it. You can Shift-click to select multiple files.

   b. **Click Add Files to List to add the file(s) to the Directory and Files pane.**

      The Load Entire Directory button inserts the "`./`" characters into the Directories and Files pane indicating that the entire workspace hierarchy be included.

5. **Type an optional comment in the Comments text field.**

   The comment is stored in the freezepoint file for future reference.

6. **Click Create.**

   A counter on the bottom left of the Freezepointing window displays the progress of the freezepoint operation.

Updating a Freezepoint File

To update a freezepoint file:

1. **In the Configuring Window, choose TeamWare ➤ Freezepointing to open the Freezepointing window.**
2. **Click the Update Freezepoint File check box.**
3. **You can accept the default name of freezepoint.out or type the name of the freezepoint file in the Freezepoint File box.**
4. **Click the browse button to the right of the workspace box or type the name of the workspace in the Workspace box.**
5. **Click Add to add files and directories to the Directories and Files list.**
6. **Type a comment in the Comments box (optional).**
7. **Click Update.**
8. **If you want to update an existing Freezepoint file, select the Update Freezepoint File check box.**

   The Use Default Workspace check box is enabled.

To freezepoint the workspace named in the existing Freezepoint File, select the Use Default Workspace check box.

Extracting Files

To extract a new source hierarchy described by a freezepoint file:

1. **In the Configuring Window, choose TeamWare ➤ Freezepointing to open the Freezepointing window.**

   The Creation tab is displayed.

2. **Click the Extraction tab.**

   The Extraction tab is displayed (see FIGURE 8-4).

3. **Type the path name of your freezepoint file in the Freezepointing File text box.**

   Path names that are not absolute are assumed to be relative to the directory in which Freezepointing is started.

4. **Click the Full Extract or the Partial Extract radio button.**

   Full Extract extracts the complete set of frozen files. Partial Extract extracts a subset that you identify. You must choose either a full or partial extract whether or not you choose to create a workspace.

5. **Click the Create Workspace check box to create a workspace that contains the SCCS histories of the frozen files.**

6. **Click one of the three Extract FreezePoint Sources radio buttons to specify the source workspace.**

   - Use default from freezepoint files--uses the path name of the source workspace as it is in the freezepoint file.
   - You specify--lets you type a workspace path name.
   - Show defaults and comments--displays the path name of the source workspace in the Workspace text box.

   To specify a source workspace hierarchy other than the one contained in the freezepoint file, click the You specify radio button and type the path name of the alternate source workspace in the Workspace text box.

7. **Type the path name of the directory in which you want the new (extracted) hierarchy to be located in the Destination Directory text box.**

   Path names that are not absolute are assumed to be relative to the directory in which Freezepointing is started.

   The destination directory that you specify can be new or existing. If you extract the hierarchy to an existing directory, you receive a warning message and must confirm the operation.

8. **Click the Extract button to begin the extraction.**

   If you selected Partial Extract, Freezepointing opens a dialog box listing the source files in the freezepoint file. Select the files you want to extract.

   Clicking the Extract button causes a series of `sccs get` operations to be performed on the source files listed in the freezepoint file. The version of each file extracted is the version specified by the SMID in the freezepoint file. The extracted g-files are written to destination directory. If you have selected Create Workspace, SCCS histories are also written to the destination directory.

   A counter on the bottom left of the Freezepointing window displays the progress of the extract operation.

FIGURE 8-3 Freezepoint in Progress

Note – If, during an extraction, Freezepointing cannot locate a file that has been renamed or deleted, the extraction is aborted and Freezepoint gives you the name of files it could not find. You must edit the freezepoint file to remove the files. Refer to the `freezepointfile` man page for information about determining the new name of a renamed file.

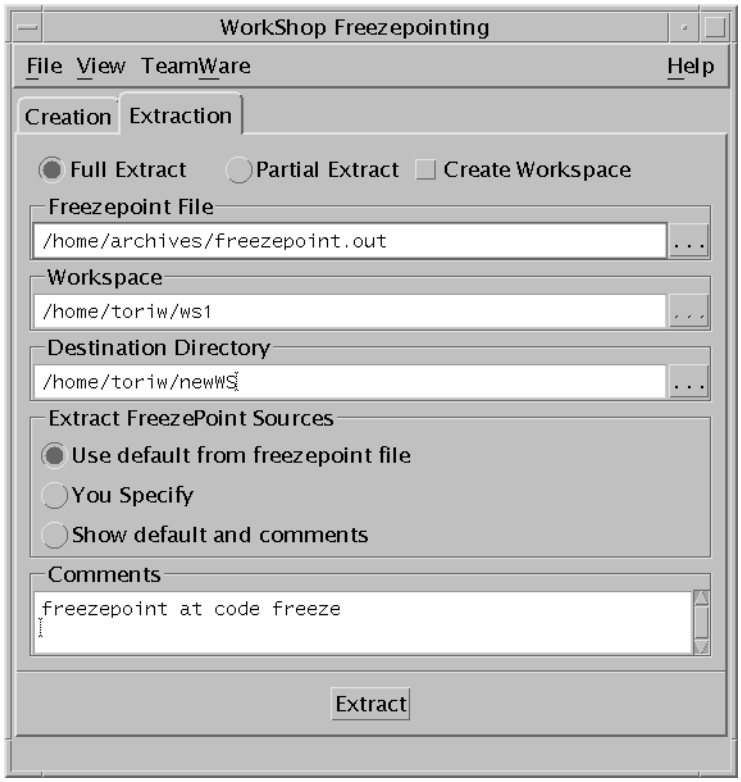


FIGURE 8-4 Freezepointing Window: Extraction Tab

TABLE 8-2 lists the items in the Freezepointing Extraction tab.

TABLE 8-2 Freezepointing Extraction Tab

Item	Description
File menu	Provides a command to exit Freezepointing.
View menu	Provides the Show Output command.
TeamWare menu	Provides commands for starting other TeamWare tools.
Tabs	Lets you switch between the Creation and Extraction tabs of the Freezepointing window.
Full Extract radio button	Lets you extract the complete set of frozen files.
Partial Extract radio button	Lets you extract a subset of the frozen files.

TABLE 8-2 Freezepointing Extraction Tab *(Continued)*

Item	Description	
Create Workspace check box	Lets you create a TeamWare workspace from the freezepoint file.	
Freezepoint File text box	Lets you type the absolute path name of the freezepoint file.	
Workspace text box	Lets you specify the source workspace.	
Extract Freezepoint radio buttons	Use default	Uses default from freezepoint file.
	You Specify	Lets you type a workspace path name.
	Show default	Uses the path name of the source workspace.
Destination Directory text box	Lets you specify the path name of the directory in which you want the new hierarchy to be located.	

Automatically Generating Freezepoints

You can configure Sun WorkShop TeamWare to create a freezepoint every time you perform a certain transactions, such as bringovers and putbacks.

To set an auto-freezepoint:

1. **In the Configuring Window, select a workspace.**
2. **Choose Workspace ➤ Properties.**
3. **Click the Freezepointing tab.**

   The Freezepointing tab of the Workspace Properties dialog box is displayed (see FIGURE 8-5).
4. **Click Yes for the time(s) you want a freezepoint created.**
5. **Click OK.**

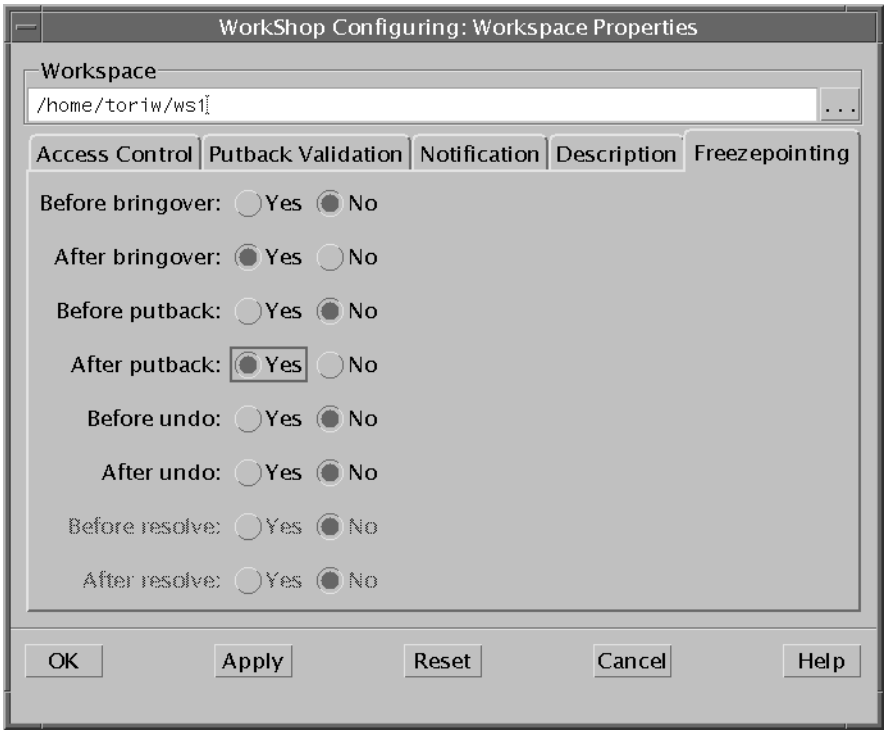


FIGURE 8-5 Workspace Properties Dialog Box: Freezepointing Tab

Freezepointing creates freezepoint files in your workspace under the `Codemgr_wsdata/Freezepoints` directory. Autofreezepoint creates a directory for each day in the format `Codemgr_wsdata/Freezepoints/YYYY/MM/DD` where `YYYY` is the year, `MM` is the month, and `DD` is the day. The freezepoint file has a name of `HHMMSS.fp.Z`, where `HH` is the hour, `MM` is the minute and `SS` is the second. The `.Z` extension indicates that autofreezepoint files are compressed with the `/bin/compress` command.

Reading Freezepoint Files Format

You can view the contents of a freezepoint file using a standard text editor.

A freezepoint file is a text file that lists the default deltas from the SCCS history files contained in the workspace hierarchy being preserved. When you later re-create the hierarchy, Freezepointing uses those entries as pointers back to the original history files and to the delta that was the default at the time the freezepoint file was created.

The deltas are not identified by their standard SCCS delta ID (SID). Instead, a new means of identification called an SCCS Mergeable ID (SMID) is used. Using the SMID enables Freezepointing to work properly with files in which SIDS have been renumbered as part of a Configuring Bringover Update transaction. For more information, see "About SCCS Mergeable IDs" on page 174.

The freezepoint file contains the following information:

- The path name of the workspace from which the list of deltas was created
- The date and time that the file was created
- The login name of the user who created the freezepoint
- 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
- An optional user-supplied comment

The following example shows a section of a freezepoint file. There are three entries; the rest of the lines are informational comments.

```
# Format:
# sfilename (previously SID) date time user hex hex hex hex hex
hex hex hex
#
# First four hex’s are the SCCS Mergeable Id of the root delta of
the
# containing delta tree.
# Last four hex’s are the SMID of the desired delta.

#sdata=99/03/10 09:20:26
#sdata=SunPro Code Manager data about conflicts, renames, etc...
#sdata=Name history : 1 0 list.c
#leaf=9f7398c4 cc06b ff6ce975 10b7656b
#leaf=f6ea91e2 bbd23cd1 3e052ed1 ca969a9e
./list.c (previously 1.5) 99/06/21 14:04:22 toriw 11db401e
cd439eeb ca3782dc 1aa255e9 97701645 bda0137e d24a3d6b 69f31f25
#sdata=99/03/10 09:20:24
#sdata=SunPro Code Manager data about conflicts, renames, etc...
#sdata=Name history : 2 1 testdir/index.html
#sdata=Name history : 1 0 index.html
#leaf=1fffddc9 85c63827 8172c838 52ba549b
./twtest/testdir/index.html (previously 1.6) 99/06/08 16:21:37
toriw 3ed3beea bb06794d 3f235871 dd89b225 d10d3db7 b8384098
3bb361a0 32e64f5e
#sdata=99/03/10 09:20:24
#sdata=SunPro Code Manager data about conflicts, renames, etc...
#sdata=Name history : 2 1 testdir/routine
#sdata=Name history : 1 0 routine
./twtest/testdir/routine1fs (previously 1.2) 99/03/12 09:11:10
toriw e59da845 d7e3b7e6 f9e7765f 43d41389 89c058c1 f758edad
81fc5a22 576e5015
```

CHAPTER 9

Building Programs in Sun WorkShop TeamWare

Sun WorkShop TeamWare lets you run one build job at a time or several build jobs concurrently. This chapter shows you how to quickly build a single application, how to customize a build, and how to fix build errors using the Building window and the Sun WorkShop editor of your choice. It covers the following topics:

- Building Window
- Building WorkShop Targets
- Building a Program
- Customizing a Build
- Fixing Build Errors

Building Window

The Building window displays information on program compilation. You can open the window by choosing TeamWare ➤ Building in the Configuring window.

Note – This is the same Building tool that you can access from Sun WorkShop. From the Sun WorkShop main window, choose Windows ➤ Show Building Window.

From the Building window, you can:

- Start a build
- Stop a build in progress
- Edit build parameters
- Save the build output to another file
- View build errors

FIGURE 9-1 shows the Building window.

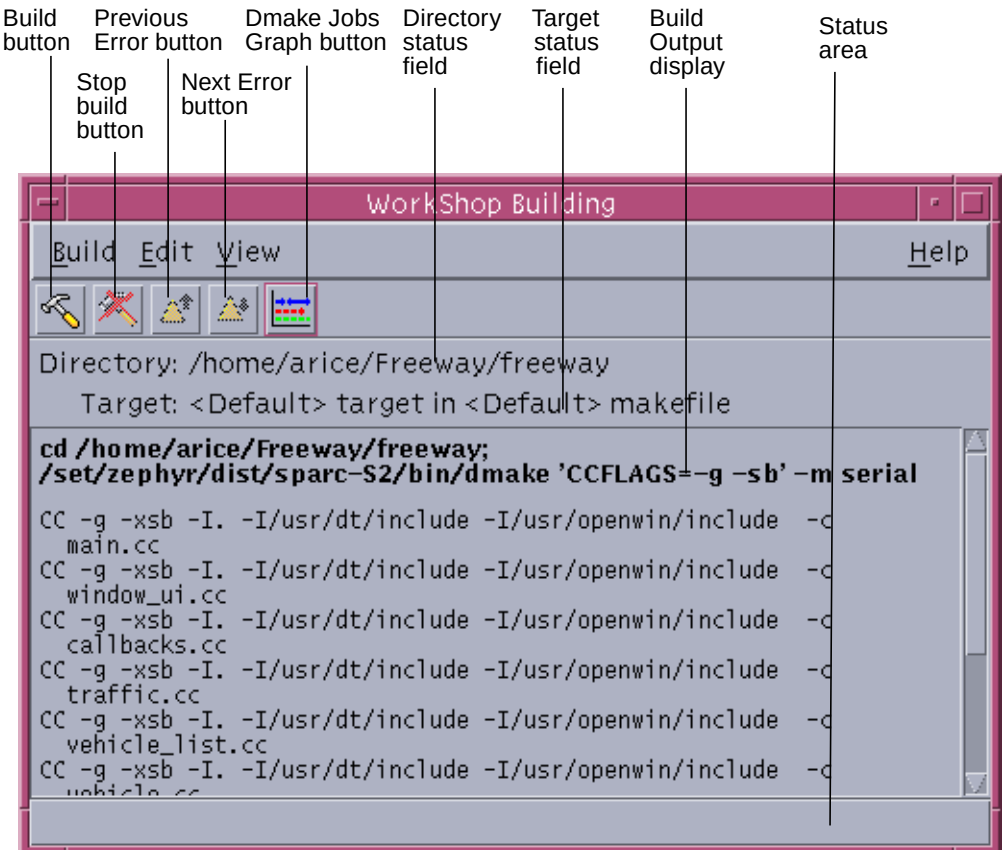


FIGURE 9-1 Building Window

TABLE 9-1 describes the components of the Building window:

TABLE 9-1 Building Window Components

Item	Description
Build menu	Provides commands to specify a target, modify the parameters of a build, start and stop a build, and save the build output.
Edit menu	Provides commands to change what is displayed in the build output display pane.
View menu	Provides commands to view build errors and to view a graphical display of builds in progress built with `dmake` and displayed in the Dmake Jobs Graph window.
Build button	Begins a build of the current Sun WorkShop target.

TABLE 9-1 Building Window Components *(Continued)*

Item	Description
Stop Build button	Stops the current build in progress.
Previous Error button	Moves the cursor to the previous build error in the Build Output Display pane and shows that error location in the text editor.
Next Error button	Moves the cursor to the next build error in the Build Output Display pane and shows that error location in the text editor.
Dmake Jobs Graph button	Opens the Dmake Jobs Graph window, which allows you to examine the build time of one or more jobs.
Directory status field	Displays the path name of the current build directory.
Target status field	Displays the name of the current `make` target.
Build Output Display pane	Provides a read-only display of build output.
Status area	Displays information about the current build.

Building WorkShop Targets

When building in the Sun WorkShop™ programming environment, two types of targets are involved: Sun WorkShop targets and user Makefile targets.

Sun WorkShop Targets

A Sun WorkShop target is an object derived from the build directory, the build command, the makefile, and the `make` target:

- *Build directory* — The directory from which the build process is invoked and also the default directory for the makefile.
- *Build command* — The command that starts the `make` utility, which reads the makefile and builds the `make` targets.
- *Makefile* — A file that contains entries that describe how to bring a `make` target up to date with respect to those files on which it depends (called *dependencies*). Because each dependency is a `make` target, it may have dependencies of its own. Targets and file dependencies and subdependencies form a tree structure that `make` traces when determining whether or not to rebuild a `make` target.

- *Make target* — An object that `make` knows how to build from the directions (rules) contained in a particular makefile. For example, a `make` target could be All or Clean. Makefiles are generally designed so that the default target (the one you get when you do not specify a target) is the most commonly built target.

When a Sun WorkShop target is built, it is added to the list of Sun WorkShop targets in the Build Menu and in the Build ➤ Edit Target command. When you begin a build, Sun WorkShop looks for the first target in the Sun WorkShop target list and builds it.

A project can contain multiple targets. For an executable, static library/archive, shared library, or Fortran application, your executable/library is one target, and a special Clean target is another (found in the Build menu picklist). The Clean target deletes all of your project's generated files (for example, the `.o` files), the source browsing database, the C++ templates database, the executable itself, and other build-related files.

For a complex project, you can have more targets that are listed in the Build menu picklist. For example, your project can generate five libraries and an executable to link them together. Each library or executable is then a WorkShop target, and you can build each individual one by selecting it from the Build menu picklist.

User Makefile Targets

A user makefile target is an object that `make` can build from the directions (rules) contained in a particular makefile. Makefiles are generally designed so that the default target (the one you get when you do not specify a target) is the most commonly built target.

A makefile contains entries that describe how to bring a `make` target up to date with respect to those files on which it depends (called dependencies). Since each dependency is a `make` target, each dependency might have dependencies of its own. Targets and file dependencies and subdependencies form a tree structure that `make` traces when deciding whether or not to rebuild a `make` target.

For a user makefile project, each target listed in the Build menu picklist is a makefile or a makefile target to be built.

Building a Program

You can begin a build without specifying a build command, makefile, or target. Or you can specify one or all of these. You can also customize a build by specifying `make` options, specifying a build mode, overriding makefile macros, or editing environment variables (see "Customizing a Build" on page 128).

Specify build parameters using the Define New Target and Edit Target dialog boxes, which are identical. You use the Define New Target dialog box to specify a new WorkShop target and the Edit Target dialog box to modify an existing WorkShop target. FIGURE 9-2 shows the Define New Target dialog box.

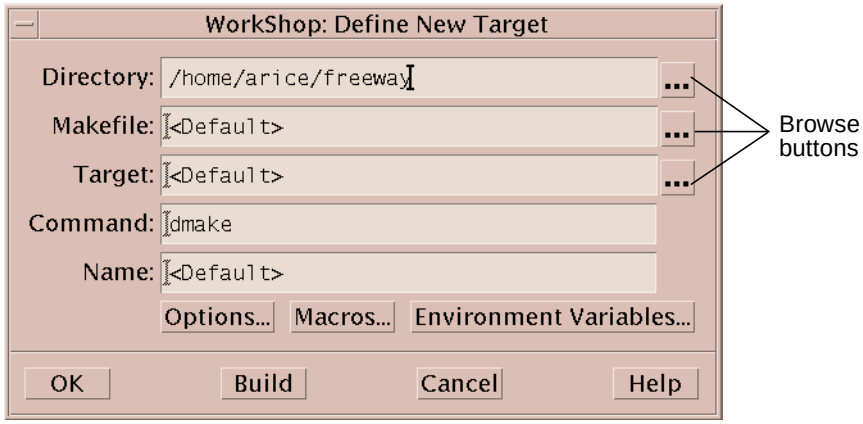


FIGURE 9-2 Define New Target Dialog Box

TABLE 9-2 describes the elements of the Define New Target and Edit Target.

TABLE 9-2 Define New Target Dialog Box

Item	Description
Directory text box	Type a build directory path, or click the browse button to open a file chooser.
Makefile text box	Type a makefile name (the default file name is makefile), or click the browse button to open a file chooser.
Target text box	Specify a target, or click on the browse button to open the Target Chooser dialog box.
Command text box	Type a `make` command. The default command is `dmake`.

TABLE 9-2 Define New Target Dialog Box *(Continued)*

Item	Description
Name text box	Allows you to assign a name to this Directory, Makefile, Target triple. The name is only used to describe the target on the menu picklist. If you do not assign a name to the target, it will appear as `<target target in <directory:<makefile makefile` on the menu picklist. Assigning a name allows you to distinguish between build targets, useful when there are several build targets in the same directory and makefile.
Options button	Lets you to modify the parameters of a build.
Macros button	Lets you to add, change, override, or delete macros to be passed into the build.
Environment Variables button	Lets you add, change, or delete environment variables to be passed into the build.
OK button	Applies the build parameters and closes the dialog box.
Build button	Applies the build parameters and builds the target.
Cancel button	Closes the dialog box without applying changes.
Help button	Displays online help about this dialog box.

Building With Default Values

Sun WorkShop provides a default make target and a default `make` command (`dmake`), so you can begin a build without specifying a build command or a `make` target. You must still supply a makefile when you are building a user makefile project or when a project is not loaded (Sun WorkShop searches for a file named makefile or Makefile and allows `make` to figure out which one to use). By using the project feature of Sun WorkShop, you can ask Sun WorkShop to create a makefile for you through the Create New Project wizard or the Edit Current Project window. For more information, see "Building With Default Values" in the Building Programs section of the online help.

Sun WorkShop offers the `dmake` build command, which parses your makefiles, determines which targets can be built concurrently, and distributes the build of those targets. `dmake` runs in one of these build modes (to set the mode, see "Customizing a Build" on page 128):

- *Serial mode* – `dmake` executes one job at a time on the local host (similar to `make`).
- *Parallel mode* – `dmake` executes multiple jobs concurrently on the local host.
- *Distributed mode* – `dmake` executes multiple jobs over several build servers. In distributed mode, you can concurrently distribute over several servers the process of building large projects consisting of many programs. `dmake` parses your

makefiles, determines which targets can be built concurrently, and distributes the build for those targets over build servers designated by you. To run in distributed mode, see "Customizing a Build" on page 128.

If you are running `dmake` in any mode, you can use the Jobs Graph window to monitor the progress of the `dmake` run and to view the state of each build job. To open the Jobs Graph window from the Building window, choose View ➤ Dmake Jobs Graph.

The Define New Target/Edit Target dialog box shows the default build values. If you do not specify a particular makefile or `make` target, Sun WorkShop looks for a file named `makefile` in the build directory and uses the first `make` target in that makefile. However, if `make` finds an SCCS history file (`s.makefile`) that is newer than the file named `makefile`, Sun WorkShop uses the most recent version of `s.makefile`. If `makefile` does not exist, Sun WorkShop searches for a file named `Makefile`.

To build a program using default build values:

1. **Look in the Directory status field in the Building window to be sure you have the correct build directory set.**

   The build directory is the directory from which the build process is run and the default directory for the makefile. If no build directory is displayed in the Directory status field or you want to change build directories, choose Build ➤ New Target to open the Define New Target dialog box. Type the build path in the Directory text box.

2. **Start a build by choosing Build ➤ Start Build.**

   The build output is displayed in the Build Output display pane in the Building window. To stop the build, click the Stop Build button in the Building window or choose Build ➤ Stop Build.

Note – The next time you open the Building window, the build directory is set to the directory in which you ran your previous build job. The path name is listed in the Directory status field.

Specifying Your Own Build Values

If you have a makefile with a unique name, a certain `make` target, or a specific build command, you can define those build values in the Define New Target dialog box or Edit Target dialog box (this applies to a user makefile project or when a project is not loaded). For example, by specifying your own build command, you can filter out unnecessary warnings by passing `make` output through a filter. At a minimum, you must include a build directory. Building will then use the `make` command to find the

makefile using `make's` search order. For more information, see "Specifying Your Own Build Values" in the Building Programs section of the online help and the `make`(1) man page.

To specify your own build values:

1. **In the Directory text box, type the name of the directory in which you want to build.**

   If you do not specify a build directory, Sun WorkShop tries to build in the directory currently displayed. If no directory is displayed, Sun WorkShop displays an error message pop-up window.

2. **In the Makefile text box, type the name of the makefile you want.**

3. **In the Target text box, type the name of the `make` target you want.**

4. **In the Command text box, type the name of the build command you want.**

   If the build command you specify is something other than `make` or `dmake`, you can specify the command and any of its arguments in the Command text box. If the path to the build command is not in your `PATH` environment variable, you might have to specify the full command path.

   The build command is formed by prepending `setenv` commands for any environment variables specified through the Environment Variables dialog box and by appending any of the `make` options specified through the Make Options and Make Macros dialog boxes.

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

   The build output is displayed in the Build Output display pane in the Building window. To stop the build, click the Stop Build button in the Building window or choose Build ➤ Stop Build.

Editing an Existing WorkShop Target

To edit an existing WorkShop target:

1. **Choose Build ➤ Edit Target in the Building window.**

2. **Choose a WorkShop target from the list.**

   The Edit Target dialog box opens, displaying the current settings for the build directory, makefile, `make` target, and build command.

3. **Edit any of the fields in the dialog box.**

   For more information, see "Specifying Your Own Build Values" on page 125.

4. **Click Build to rebuild the WorkShop target with your new settings.**

Collecting Build Output

Build output is cleared automatically from the Build Output display pane in the Building window each time you run a build job.

To accumulate output from builds:

1. **Choose Edit ➤ Accumulate Output.**

   The output for the subsequent build is displayed below the output for the previous build.

2. **Choose Build ➤ Build.**

3. **Scroll through the build output display pane to see the output for each build.**

   Each build job output begins with the build path and the name of the build target.

   To clear the build output pane, choose Edit ➤ Clear Results.

Saving Build Output

You can maintain a history of build output information for one or more build jobs by saving to a file the output in the build output display pane of the Building window.

To save build output:

1. **In the Building window, choose Build ➤ Save Output As.**

2. **Select or create a file in which to save the build output.**

   The build output log is saved as a text file.

Removing a WorkShop Target

You can remove targets from the Edit Target list in the Build menu.

To remove a target:

1. **Choose Build ➤ Remove Targets From Menu in either the Sun WorkShop main window or the Building window.**

2. **Select one or more targets from the list in the Remove targets from menu dialog box.**

   Press the Control key and click to select more than one target name.

3. **Click OK.**

Customizing a Build

You can customize a build by changing `make` options, specifying a build mode, using makefile macros, or using environment variables. Choose Build ➤ Edit Target and choose a target from the list. The Edit Target dialog box opens.

Specifying Build Options

You can specify build options in the Build Options dialog box. To open the Build Options dialog box, click Options in the Edit Target dialog box. When you are finished selecting the options you want, click OK in the Build Options dialog box. Then click Build in the Edit Target dialog box.

See the `dmake` and `make` man pages for detailed information about the following commands and options:

Category: Basic

- Displays commands but does not run them (`-n`)
- Continues with dependency branches that do not depend on the target when an error occurs or when `make` cannot find a rule (`-k`)

Category: Execute Commands and Display

- Displays the reasons why `make` rebuilds a target (`-d`)

  `make` displays any and all dependencies that are newer. The `make` display options are also read in from the `MAKEFLAGS` environment variable.
- Displays detailed information on the dependency check and processing (`-dd`)
- Displays the text of the makefiles read in (`-D`)
- Displays the text of the makefiles, `make.rules` file, the state file, and all hidden-dependency reports (`-DD`)
- Executes commands without echoing them (`-s`). This option is equivalent to the special function target `.SILENT:`.

Category: Display Instead of Executing

- Prints the complete set of macro definitions and target descriptions (`-p`)
- Reports dependencies only; does not build them (`-P`)
- Returns a zero or nonzero status code depending on whether or not the target file is up to date (`-q`)

Category: Miscellaneous

- Touches the target files (updates them) instead of performing their rules (`-t`)
- Ignores the default makefile `/usr/share/lib/make/make.rules` (`-r`)
- Allows environment variables to override assignments within makefiles (`-e`)
- Ignores error codes returned by commands (`-i`). This option is equivalent to the special function target `.IGNORE:`.

Category: Distributed Make

This category allows you to specify the type of `make` process to run. TABLE 9-3 describes the `dmake` options and the actions you need to take for each Mode you select.

TABLE 9-3 `dmake` Options

Item	Action	
Mode	Select the type of `make` process to run:	
	Serial	Do not fill in any text boxes.
	Parallel	Specify the maximum number of build jobs to be run in the Maximum Jobs text box.
	Distributed	1. Specify the maximum number of build jobs to be run in the Maximum Jobs text box. 2. Specify the name or path of the `.dmakerc` file in the Runtime Configuration File text box. 3. Specify a Build server group in the Build server group text box. 4. Type the path name for your preferred output directory in the Temporary output directory text box.
Maximum jobs	Type the maximum number of jobs that are distributed to the build servers. `dmake` uses the sum of the jobs specified for the servers in the group if you do not specify a number.	

TABLE 9-3 `dmake` Options *(Continued)*

Item	Action
Runtime configuration File	Specify a runtime configuration file. `dmake` uses the default of `~/.dmakerc`.
Build server Group	Type the name of the server group to which jobs are distributed. `dmake` uses the default of the first group listed in the `.dmakerc` file.
Temporary output directory	Specify the name of the directory to which temporary output is to be written. `dmake` uses `~/.dmake` as the default.

Before running a distributed build for the first time, you must create a `.dmakerc` runtime configuration file that specifies which machines are to participate as `dmake` build servers. The file contains lists of build servers and the number of jobs distributed to each build server. The `dmake` utility searches for this file on the `dmake` host to know where to distribute jobs. Generally, this file is in your home directory. If `dmake` does not find a runtime configuration file, it distributes two jobs to the local host. For information on setting up a runtime configuration file, see the `dmake`(1) man page.

Before a machine can be used as a build server, it must be configured to allows jobs to be distributed to it. A build server should be of the same architecture and running the same operating system version as the `dmake` host. Be default, it is assumed that the path to the `dmake` executables is the same for the `dmake` host as it is for the build server. If it is not, you must customize the path attribute for that server.

To set up a machine to be used as a build server, you must create a configuration file called `/etc/opt/SPROdmake/dmake.conf` on the server's file system. Without this file, `dmake` refuses to distribute jobs to that machine. In the `dmake.conf` file, you specify the maximum number of jobs (from all users) that can run concurrently on that build server. For more information on `dmake`, see Chapter 10 and the `dmake`(1) man page.

Using Makefile Macros

You can specify makefile macros in the Make Macros dialog box. Makefile macros let you refer conveniently to files or command options that appear in the description file. Through the Make Macros dialog box, you can add makefile macros to or delete them from the Persistent Build Macros list in your WorkShop target, and then reassign values for makefile macros in the list. You can also add macros currently defined in the makefile to the list and override their values. For information on defining macros, see *Introduction to Sun WorkShop*, Appendix B.

Adding a Macro

To add a macro to the Persistent Build Macros list:

1. **Click Macros in the Edit Target dialog box.**
2. **Type the name of a macro in the Name text box.**
3. **Type a value (or definition) for the macro in the Value text box.**
4. **Click Add to add the new macro to the list.**
5. **Repeat the previous three steps to add other macros.**
6. **Click OK to close the dialog box.**

Deleting a Macro

To delete a macro from the Persistent Build Macros list:

1. **Select a macro in the Persistent Build Macros list in the Make Macros dialog box.**
2. **Click Delete (Delete All removes all macros in the list).**
3. **Click OK to establish the change and close the dialog box.**

Changing a Macro

To change the value of a macro (what the macros name actually represents) in the Persistent Build Macros list:

1. **Click Macros in the Edit Target dialog box.**
2. **Click More in the Make Macros dialog box.**
3. **Select a macro in the Makefile Macros list.**
4. **Click <<Add to add the macro to the Persistent Build Macros list.**
5. **Type a new value in the Value text box.**
6. **Click Change.**
7. **Click OK to establish the change and close the dialog box.**
8. **Click Build in the Edit Target dialog box to start the build with the new values.**

Reviewing and Overriding Makefile Macros

A macro definition that appears in the Persistent Build Macros list overrides any macro with the same name that appears in the makefile.

To review the current macro definitions, click More to open the Makefile Macros list, which displays all the macros that are defined in the makefile associated with the build target. You can filter the list using the Filter text box.

To override the value of a makefile macro:

1. **Select a macro in the Makefile Macros list.**
2. **Click <<Add to add the macro to the Persistent Build Macros list.**
3. **Type a new value in the Value text box.**
4. **Click Change.**
5. **Click OK to establish the change and close the dialog box.**

   The macro definition in the Persistent Build Macros list overrides the macro definition in the makefile.
6. **Click Build in the Edit Target dialog box to start the build with the new values.**

Using Environment Variables

You can specify environment variables for your build in the Environment Variables dialog box. Using the Environment Variables dialog box, you can add environment variables to or delete them from the Persistent Environment Variables list in your WorkShop target and reassign values for environment variables in the list. When you start the build, `setenv` commands for these environment variables are prepended to the build command.

Adding an Environment Variable

To add an environment variable to the Persistent Environment Variables list:

1. **Click Environment Variables in the Edit Target dialog box.**
2. **Type the name of an environment variable in the Name text box.**
3. **Type a value for the variable in the Value text box.**
4. **Click Add to add the environment variable to the Persistent Environment Variables list.**

5. **Repeat Step 2 through Step 4 to add other environment variables.**
6. **Click OK to close the dialog box.**

Deleting an Environment Variable

To delete a variable from the Persistent Environment Variables list:

1. **Select a variable from the list.**
2. **Click Delete (Delete All removes all environment variables in the list).**
3. **Click OK to establish the change and close the dialog box.**

Changing the Value of an Environment Variable

To change the value of an environment variable in the Persistent Environment Variables list:

1. **Select an environment variable in the list.**
2. **Type a new value in the Value text box and click Change.**
3. **Click OK to establish the change and close the dialog box.**
4. **Click Build to start the build with the new build environment.**

Reviewing and Overriding Environment Variables

An environment variable definition that appears in the Persistent Environment Variables list overrides any environment variable with the same name that appears in the current Building process environment. To review the current Building process environment variable definitions, click More to open the Current Environment list, which includes all the environment variables that are currently defined in the Building process environment. You can filter the list using the Filter text box.

To override the value of an environment variable:

1. **Select an environment variable in the Current Environment list.**
2. **Click <<Add to add the environment variable to the Persistent Environment Variables list.**
3. **Type a new value in the Value text box and click Change.**
4. **Click OK to establish the change and close the dialog box.**
5. **Click Build in the Edit Target dialog box to start the build with the new values.**

Fixing Build Errors

The process of fixing build errors is simplified by the integration of the text editor into the build process. When a build fails, the build errors are displayed in the Build Output display pane of the Building window, as shown in FIGURE 9-3. Build errors have hypertext links (highlighted and underscored) to the source files containing the errors. Clicking on the underscored error in the Building window starts a text editor that displays the source file containing the error.

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

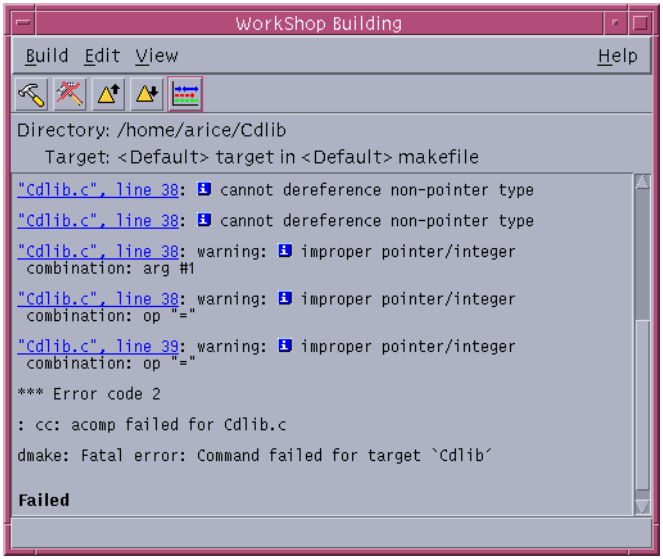


FIGURE 9-3 Build Errors in the Build Output Display Pane

Error messages issued by the C compiler include an icon ( ) in the build error message. Click on the icon to open a dialog box that defines the associated error message.

Note – Only Sun compilers produce output that can be converted to hypertext links. If you use a build command that does not call Sun compilers, you will not have links to the source files from the build errors listed in the Building window.

Displaying the Source of an Error

When you click on the underscored error in the Building window, your text editor opens and displays the source file containing the error. The source file is shown with the error line highlighted and an error icon appears to the left of the line (see FIGURE 9-4).

Use the keyboard shortcuts F4 (next error) and Shift+F4 (previous error) to navigate through the build errors so you can keep a focus on the text editor window.

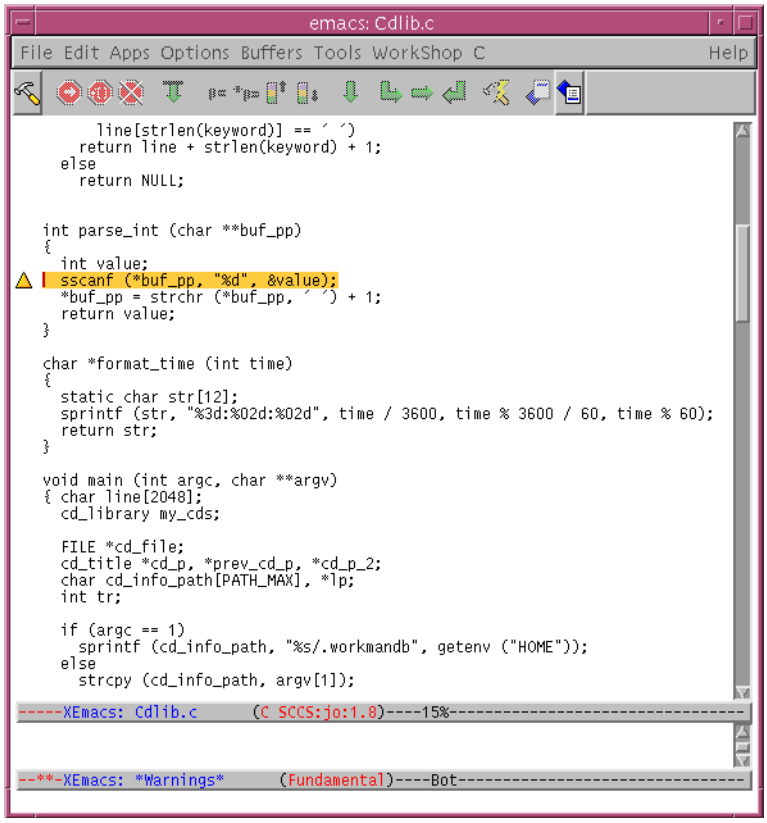


FIGURE 9-4 Text Editor Window Displaying Source File With Error

Fixing an Error

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

1. **Click a highlighted error in the Build Output display pane.**

   The editor window opens, displaying the source file containing the error with the cursor positioned at the error line. Yellow highlight indicates the current error.

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

3. **To view another error, click the Next Error button in the tool bar (or use the keyboard shortcut F4) to go to the location of the next build error in the text editor.**

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

4. **Save the edited file.**

5. **Click the Build button in the text editor's tool bar to rebuild.**

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

Exiting Building

To kill the current build process and close all build windows, choose Build ➤ Exit Building in the Building window.

CHAPTER 10

Using the dmake Utility

This chapter describes the way the distributed make (dmake) utility distributes builds over several hosts to build programs concurrently over a number of workstations or multiple CPUs.

- Basic Concepts
- Understanding the dmake Utility
- Impact of the dmake Utility on Makefiles
- Using the dmake Utility

Basic Concepts

Distributed make (dmake) allows you to concurrently distribute the process of building large projects, consisting of many programs, over a number of workstations and, in the case of multiprocessor systems, over multiple CPUs. The dmake utility parses your makefiles and:

- Determines which targets can be built concurrently
- Distributes the build of those targets over a number of hosts

The dmake utility is a superset of the make utility.

To understand dmake, you should know about:

- Configuration files (runtime and build server)
- The dmake host
- The build server

Configuration Files

The dmake utility consults two files to determine to which build servers jobs are distributed and how many jobs can be distributed to each.

Runtime Configuration File

The dmake utility searches for a runtime configuration file on the dmake host to know where to distribute jobs. Generally, this file is in your home directory on the dmake host and is named .dmakerc. It consists of a list of build servers and the number of jobs to be distributed to each build server. See "The dmake Host" section for more information.

Build Server Configuration File

Each build server that you want to participate in a distributed build must have a /etc/opt/SPROdmake/dmake.conf file. This file specifies the maximum total number of dmake jobs that can be distributed to this build server by all dmake users. In addition, it may specify the "nice" priority under which all dmake jobs should run.

See "The Build Server" on page 141 for more information.

The dmake Host

The dmake host is defined as the machine on which the dmake command is initially invoked. The dmake utility searches for a runtime configuration file to determine where to distribute jobs. Generally, this file must be in your home directory on the dmake host and is named .dmakerc. The dmake utility searches for the runtime configuration file in these locations and in the following order:

1. The path name you specify on the command line using the -c option
2. The path name you specify using the DMAKE_RCFILE makefile macro
3. The path name you specify using the DMAKE_RCFILE environment variable
4. $(HOME)/.dmakerc

If a runtime configuration file is not found, the dmake utility distributes two jobs to the dmake host.

The runtime configuration file allows you to specify a list of build servers and the number of jobs you want distributed to each build server. The following is an example of a `.dmakerc` file:

CODE EXAMPLE 10-1 `dmakerc` file

```
# My machine. This entry causes dmake to distribute to it.
falcon  { jobs = 1 }
hawk
eagle   { jobs = 3 }
# Manager's machine. She's usually at meetings
heron   { jobs = 4 }
avocet
```

- The entries: `falcon`, `hawk`, `eagle`, `heron`, and `avocet` are listed build servers.
- You can specify the number of jobs you want distributed to each build server. The default number of jobs is two.
- Any line that begins with the # character is interpreted as a comment.

Note – In the code example above, list of build servers includes `falcon` which is also the `dmake` host. The `dmake` host can also be specified as a build server. If you do not include it in the runtime configuration file, no `dmake` jobs are distributed to it.

You can also construct groups of build servers in the runtime configuration file. `dmake` provides you with the flexibility of easily switching between different groups of build servers as circumstances warrant. For instance, you may define groups of build servers for builds under different operating systems, or you may define groups of build servers that have special software installed on them.

The following is an example of a runtime configuration file that contains groups of build servers:

```
earth  { jobs = 2 }
mars   { jobs = 3 }

group lab1 {
       host falcon{ jobs = 3 }
       host hawk
       host eagle { jobs = 3 }
}

group lab2 {
       host heron
       host avocet{ jobs = 3 }
       host stilt { jobs = 2 }
}

group labs {
       group lab1
       group lab2
}

group sunos5.x {
       group labs
       host jupiter
       host venus{ jobs = 2 }
       host pluto { jobs = 3 }
}
```

- Formal groups are specified by the `group` keyword and lists of their members are delimited by braces ({}).
- Build servers that are members of groups are specified by the optional `host` directive.
- Groups can be members of other groups.
- Individual build servers can be listed in runtime configuration files that also contain groups of build servers; in this case, `dmake` treats these build servers as members of the *unnamed* group.

In order of precedence, the `dmake` utility distributes jobs to the following:

1. The formal group specified on the command-line as an argument to the `-g` option
2. The formal group specified by the `DMAKE_GROUP` makefile macro
3. The formal group specified by the `DMAKE_GROUP` environment variable

4. The first group specified in the runtime configuration file

The dmake utility allows you to specify a different execution path for each build server. By default dmake looks for the dmake support binaries on the build server in the same logical path as on the dmake host. You can specify alternate paths for build servers as a host attribute in the .dmakerc file. For example:

```
group lab1 {
       host falcon{ jobs = 10 , path = "/set/dist/sparc-S2/bin" }
       host hawk{ path = "/opt/SUNWspro/bin"                    }
}
```

You can use double quotation marks to enclose the names of groups and hosts in the .dmakerc file. This allows you more flexibility in the characters that you can use in group names. Digits are allowed, as well as alphabetic characters. Names that start with digits should be enclosed in double quotes. For example:

```
group "123_lab" {
       host "456_hawk"{ path = "/opt/SUNWspro/bin"              }
}
```

The Build Server

The /etc/opt/SPROdmake/dmake.conf file is in the file system of build servers. Use this file to limit the maximum number of dmake jobs (from all users) that can run concurrently on a build server and to specify the "nice" priority under which all dmake jobs should run. The following is an example of an /etc/opt/SPROdmake/dmake.conf file. This file sets the maximum number of dmake jobs permitted to run on a build server (from all dmake users) to be eight.

```
max_jobs: 8
nice_prio: 5
```

Note – If the /etc/opt/SPROdmake/dmake.conf file does not exist on a build server, no dmake jobs will be allowed to run on that server.

Understanding the dmake Utility

To run a distributed make, use the executable file dmake in place of the standard make utility. You should understand the Solaris make utility before you use dmake. If you need to read more about the make utility, see the Solaris *Programming Utilities Guide*. If you use the make utility, the transition to dmake requires little if any alteration.

Impact of the dmake Utility on Makefiles

The methods and examples in this section show the kinds of problems that dmake can help solve. As procedures become more complicated, so do the makefiles that implement them. You must know which approach will yield a reasonable makefile that works. The examples in this section illustrate common code-development predicaments and some straightforward methods to simplify them using dmake.

Using Makefile Templates

If you use a makefile template from the outset of your project, custom makefiles that evolve from the makefile templates will be:

- More familiar
- Easier to understand
- Easier to integrate
- Easier to maintain
- Easier to reuse

The less time you spend editing makefiles, the more time you have to develop your program or project.

Building Targets Concurrently

Large software projects typically consist of multiple independent modules that can be built concurrently. The dmake utility supports concurrent processing of targets on multiple machines over a network. This concurrency can markedly reduce the time required to build a large project.

When given a target to build, `dmake` checks the dependencies associated with that target and builds those that are out of date. Building those dependencies may, in turn, entail building some of their dependencies. When distributing jobs, `dmake` starts every target that it can. As these targets complete, `dmake` starts other targets. Nested invocations of `dmake` are not run concurrently by default, but this can be changed (see "Restricting Parallelism" on page 146 for more information).

Since `dmake` builds multiple targets concurrently, the output of each build is produced simultaneously. To avoid intermixing the output of various commands, `dmake` collects output from each build separately. The `dmake` utility displays the commands before they are executed. If an executed command generates any output, warnings, or errors, `dmake` displays the entire output for that command. Since commands started later may finish earlier, this output may be displayed in an unexpected order.

Limitations on Makefiles

Concurrent building of multiple targets places some restrictions on makefiles. Makefiles that depend on the implicit ordering of dependencies may fail when built concurrently. Targets in makefiles that modify the same files may fail if those files are modified concurrently by two different targets. Some examples of possible problems are discussed in this section.

Dependency Lists

When building targets concurrently, it is important that dependency lists be accurate. For example, if two executables use the same object file but only one specifies the dependency, then the build may cause errors when done concurrently. For example, consider the following makefile fragment:

```
all: prog1 prog2
prog1: prog1.o aux.o
    $(LINK.c) prog1.o aux.o -o prog1
prog2: prog2.o
    $(LINK.c) prog2.o aux.o -o prog2
```

When built serially, the target `aux.o` is built as a dependent of `prog1` and is up-to-date for the build of `prog2`. If built in parallel, the link of `prog2` may begin before `aux.o` is built, and is therefore incorrect. The `.KEEP_STATE` feature of `make` detects some dependencies, but not the one shown above.

Explicit Ordering of Dependency Lists

Other examples of implicit ordering dependencies are more difficult to fix. For example, if all of the headers for a system must be constructed before anything else is built, then everything must be dependent on this construction. This causes the makefile to be more complex and increases the potential for error when new targets are added to the makefile. The user can specify the special target `.WAIT` in a makefile to indicate this implicit ordering of dependents. When `dmake` encounters the `.WAIT` target in a dependency list, it finishes processing all prior dependents before proceeding with the following dependents. More than one `.WAIT` target can be used in a dependency list. The following example shows how to use `.WAIT` to indicate that the headers must be constructed before anything else.

```
all: hdrs .WAIT libs functions
```

You can add an empty rule for the `.WAIT` target to the makefile so that the makefile is compatible with the `make` utility.

Concurrent File Modification

You must make sure that targets built concurrently do not attempt to modify the same files at the same time. This can happen in a variety of ways. If a new suffix rule is defined that must use a temporary file, the temporary file name must be different for each target. You can accomplish this by using the dynamic macros `$@` or `$*`. For example, a `.c.o` rule that performs some modification of the `.c` file before compiling it might be defined as:

```
.c.o:
    awk -f modify.awk $*.c > $*.mod.c
    $(COMPILE.c) $*.mod.c -o $*.o
    $(RM) $*.mod.c
```

Concurrent Library Update

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

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

A better method is to build each object file and then archive all the object files after completion of the builds. An appropriate suffix rule and the corresponding library rule are:

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

Multiple Targets

Another form of concurrent file update occurs when the same rule is defined for multiple targets. An example is a `yacc`(1) program that builds both a program and a header for use with `lex`(1). When a rule builds several target files, it is important to specify them as a group using the + notation. This is especially so in the case of a parallel build.

```
y.tab.c y.tab.h: parser.y
    $(YACC.y) parser.y
```

This rule is actually equivalent to the two rules:

```
y.tab.c: parser.y
    $(YACC.y) parser.y
y.tab.h: parser.y
    $(YACC.y) parser.y
```

The serial version of `make` builds the first rule to produce `y.tab.c` and then determines that `y.tab.h` is up-to-date and need not be built. When building in parallel, `dmake` checks `y.tab.h` before `yacc` has finished building `y.tab.c` and notices that `y.tab.h` *does* need to be built, it then starts another `yacc` in parallel with the first one. Since both `yacc` invocations are writing to the same files (`y.tab.c` and `y.tab.h`), these files are apt to be corrupted and incorrect. The correct rule uses the + construct to indicate that both targets are built simultaneously by the same rule. For example:

```
y.tab.c + y.tab.h: parser.y
    $(YACC.y) parser.y
```

Restricting Parallelism

Sometimes file collisions cannot be avoided in a makefile. An example is `xstr`(1), which extracts strings from a C program to implement shared strings. The `xstr` command writes the modified C program to the fixed file `x.c` and appends the strings to the fixed file `strings`. Since `xstr` must be run over each C file, the following new `.c.o` rule is commonly defined:

```
.c.o:
    $(CC) $(CPPFLAGS) -E $*.c | xstr -c -
    $(CC) $(CFLAGS) $(TARGET_ARCH) -c x.c
    mv x.o $*.o
```

The `dmake` utility cannot concurrently build targets using this rule since the build of each target writes to the same `x.c` and `strings` files. Nor is it possible to change the files used. You can use the special target `.NO_PARALLEL`: to tell `dmake` not to build these targets concurrently. For example, if the objects being built using the `.c.o` rule were defined by the `OBJECTS` macro, the following entry would force `dmake` to build those targets serially:

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

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

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

Nested Invocations of Distributed Make

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

If you nest commands that you know will not collide, you can force them to be done in parallel by using the `.PARALLEL:` construct.

When a makefile contains many nested commands that run concurrently, the load-balancing algorithm may force too many builds to be assigned to the local machine. This may cause high loads and possibly other problems, such as running out of swap space. If such problems occur, allow the nested commands to run serially.

Using the `dmake` Utility

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

- From the `dmake` host (the machine you are using) you must be able to use `rsh`, without being prompted for a password, to remotely execute commands on the build server. See the `rsh`(1) man page for more information about the `rsh` command. For example:

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

- The `bin` directory in which the `dmake` software is installed must be accessible from the build server. It is common practice to have all build servers share a common `dmake` installation directory. See the `share`(1M) and `mount`(1M) man pages or the system AnswerBook documentation for more information about creating shared filesystems.

- By default, dmake assumes that the logical path to the dmake executables on the build server is the same as on the dmake host. You can override this assumption by specifying a path name as an attribute of the host entry in the runtime configuration file. For example:

```
group sparc-cluster {
   host wren   { jobs = 10 , path = "/export/SUNWspro/bin"}
   host stimpy { path = "/opt/SUNWspro/bin"            }
```

- The source hierarchy you are building must be:
  - Accessible from the build server
  - Mounted under the same name

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

If you specify the -m option with the parallel argument, or set the DMAKE_MODE variable or macro to the value parallel, dmake does not scan your runtime configuration file. Therefore, you must specify the number of jobs using the -j option or the DMAKE_MAX_JOBS variable or macro. If you do not specify a value this way, a default of two jobs is used.

If you modify the maximum number of jobs using the -j option, or the DMAKE_MAX_JOBS variable or macro when using dmake in distributed mode, the value you specify overrides the values listed in the runtime configuration file. The value you specify is used as the total number of jobs that can be distributed to all build servers.

If you access dmake from the Building window, use the online help to see how to specify your build servers and jobs. If you access dmake from the command line, see the dmake man page (dmake.1).

CHAPTER 11

Sun WorkShop TeamWare Shortcuts

Sun WorkShop TeamWare provides several shortcuts to make it easier to use. This chapter discusses the following topics:

- Accessing TeamWare From the Command Line
  - Configuring Commands
  - Versioning Commands
  - Merging Commands
  - Freezepoint Commands
- GUI Shortcuts

Accessing TeamWare From the Command Line

In addition to working with Sun WorkShop TeamWare tools through the graphical user interface (GUI), you can also type TeamWare commands at any system prompt. You can access Sun WorkShop TeamWare commands through the GUI and at the command line, and both can be used interchangeably. You can simultaneously use the GUI for some functions and work at the command line for others. You can include these commands in scripts or programs to further automate your file management.

Configuring Commands

To see a list of Configuring commands, type `codemgr`.

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

This list includes the syntax of each command with command options enclosed in square brackets. For example, here is the syntax of the `bringover` command:

```
bringover [-w child_ws] [-p parent_ws] [-c comment]
[-m comment_file] [[-f flp]...] [-n] [-g] [-q] [-v] [-B] [-C]
[files and dirs]
```

Here's how you could issue a `bringover` command at the command line:

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

Sun WorkShop TeamWare tools provide several ways to reduce typing long command lines, including environment variables and argument files that store previously specified arguments. To get a detailed listing of the command and the options you can use with the command, type `man <commandname>`, for example:

```
% man bringover
```

Note – Because Sun WorkShop product man pages do not install into the standard `/usr/share/man` directories, you must change your `MANPATH` environment variable to enable access to Sun WorkShop TeamWare man pages. See "Accessing Sun WorkShop Documentation" on page 6.

TABLE 11-1 lists often-used Configuring menu items and their corresponding commands.

TABLE 11-1 Configuring Menu Items and Corresponding Commands

GUI Menu Item	Corresponding Command
Create Workspace	`workspace create`
Rename	`workspace move`
Parent	`workspace parent`
Bringover Create	`bringover`
Bringover Update	`bringover`
Putback	`putback`
Undo	`ws_undo`
Resolve	`resolve`

To make a workspace the default for command line commands, set the `CODEMGR_WS` environment variable. See "Configuring Environment Variables" on page 51.

Versioning Commands

To start Versioning from the command line, type `twversion` followed by the ampersand (&):

```
demo% twversion &
```

Merging Commands

To start Merging from the command line without loading any input files (assuming that the Merging executable is in your search path), at a shell command prompt, type `twmerge` followed by the ampersand (&):

```
demo% twmerge &
```

The `twmerge` command starts Merging (in the background) without loading any files. To start Merging with files loaded in each window, see "Loading Two Files at Startup" on page 152.

The complete `twmerge` command is summarized below, with command options enclosed in square brackets.

```
twmerge [-b] [-r] [-tabsize number] [-diffopt [bwi]]
[-a  ancestor] [-f1  name1] [-f2  name2]
[-l listfile] [  leftfile rightfile [outfile]  ] [-V]
```

Loading Two Files at Startup

To load two files at the time you start Merging from the command line, change to the directory in which the files are stored and specify the file names on the command line. To merge two files named `file_1` and `file_2`, use the following command:

```
demo% twmerge file_1 file_2 &
```

The first file listed appears in the left text pane; the second file appears in the right pane.

Loading Three Files at Startup

To merge the same two files and at the same time compare them to a common ancestor named `ancestor_file`, change to the directory in which the files are stored and use the following command:

```
demo% twmerge -a ancestor_file file_1 file_2 &
```

The ancestor file is not displayed, but differences between the ancestor file and the two descendants are marked, and the merged output file is based on the ancestor file.

Freezepoint Commands

You can create, update, and extract freezepoints at the command line. To see a list of Freezepoint commands, type `freezept`.

```
example% freezept
        compare ...
        create ...
        diff ...
        extract ...
        help
        sid ...
        smid ...
        update  ....
```

For information about using Freezepoint from the command line, see the `freezept` man page. Type:

```
% man freezept
```

GUI Shortcuts

There are several shortcuts built into the Sun WorkShop TeamWare interface. This is true for selecting workspaces, directories and files:

- Click an icon to select a single workspace, file or directory. Shift-click to select multiple workspaces, files or directories.
- Select groups of files by clicking the left mouse button in an empty portion of the dialog box 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.
- Select files and directories by moving the pointer over any file or directory icon and clicking. You can extend the selection to include any number of additional files and directories by moving the pointer over them and clicking the middle mouse button.

TABLE 11-2 lists other Sun WorkShop TeamWare mouse and keyboard shortcuts.

TABLE 11-2 Mouse and Keyboard Shortcuts

Action	Result	For More Information
Drag and drop workspace icon on an open area.	Displays Bringover Create Transaction dialog box.	
Drag and drop parent workspace icon on a child workspace icon.	Displays Bringover Update tab in the transaction dialog box.	
Drag and drop child workspace icon on a parent workspace icon.	Displays Putback tab in the Transaction dialog box.	
Drag and drop workspace icon on an unrelated workspace icon.	Displays a pop-up dialog box asking if you want to perform a bringover or a putback.	
Control+drag and drop workspace icon on another workspace.	Reparents workspace.	"Reparenting a Workspace" on page 44.
Control+drag and drop workspace icon on an open area.	Orphans workspace.	"Reparenting a Workspace" on page 44.
Click workspace icon name field.	Renames workspace.	
Click right mouse button in Configuring.	Displays a combination menu of File, Workspace and Transaction menus.	
Press Props key in Configuring on empty area.	Displays CodeManager tab of Tool Properties dialog box.	"Customizing Configuring Using Tool Properties" on page 49.
Press Props key in Configuring with a workspace selected.	Displays Workspace Properties dialog box.	"Customizing Configuring Using Tool Properties" on page 49.
Double-click workspace icon in Configuring.	Launches a tool. User configurable, Versioning is the default.	"Double-Click Actions in Configuring" on page 155.
Double-click icon of a workspace that contains conflicts.	Launches a tool. User configurable, Resolve window is the default.	"Double-Click Actions in Configuring" on page 155.

TABLE 11-2 Mouse and Keyboard Shortcuts *(Continued)*

Action	Result	For More Information
Double-click on file icon in Versioning.	Checks out the file or displays file history; user configurable.	"Double-Click Actions in Configuring" on page 155.
Double-click on delta in Versioning File History.	Displays the delta in your default editor.	"Double-Click Actions in Configuring" on page 155.
Click right mouse button in Versioning.	Displays a combination menu of Select Files and commands menu.	
Click right mouse button in Versioning File History.	Displays a combination menu of the Version menu and Merge Branches.	"Viewing File History" on page 90.
Click right mouse button in Merging.	Displays the Navigate menu.	"Resolving Differences" on page 77.

Double-Click Actions in Configuring

When you double-click with the pointer over a workspace icon, the TeamWare Versioning is automatically started (with the selected workspace automatically loaded). If you double-click when the pointer is over the icon of a workspace that contains unresolved conflicts, Configuring automatically activates the Resolve tab. Conflicted files from the selected workspace are automatically loaded and ready for processing. You can customize Configuring double-click behavior using the Tool Properties dialog box (see FIGURE 3-7).

Double-Click Actions in Versioning

By default, double-clicking on a file in the Versioning window "toggles" the state of the file, that is, it will check out the file, or if the file is already checked out, it will check in the file. You can change the double-click action to display the file history instead in Versioning Options (View ➤ Options, General Tab). See "Changing Versioning Properties" on page 100.

Double-Click Action File History

Double-clicking on a delta in the Versioning File History window displays the delta in your default editor.

CHAPTER 12

Sun WorkShop TeamWare Architecture

This chapter describes the underlying files that Sun WorkShop TeamWare uses to track workspaces and files. It also describes the ways Configuring manipulates SCCS history files when you copy files between workspaces and resolve conflicts.

This chapter discusses the following topics:

- Workspace Metadata Directory
- Configuring Defaults Files
- How Configuring Merges Files
- About SCCS Mergeable IDs

Workspace Metadata Directory

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

Configuring provides the tools necessary to maintain the information kept in the `Codemgr_wsdata` directory. Although not recommended, it might be necessary to modify certain files manually; however, you must be very careful to preserve the

format of each file you edit. TABLE 12-1 briefly describes each of the files and directories contained in the `Codemgr_wsdata` directory. Information regarding the format of these files is available in the `man`(4) page for each file.

TABLE 12-1 Contents of the `Codemgr_wsdata` Metadata Directory

File/Dir Name	Description
`access_control`	Contains information that controls which users are allowed to execute Configuring transactions and commands for a given workspace. When workspaces are created, a default access control file is also created. See Chapter 4 and "The access_control File" on page 160.
`args`	Contains a list of file, directory, and FLP arguments and is maintained by the Configuring, Bringover, and Putback transaction commands. Initially, the `args` 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 `args` file; if they are, the new arguments replace the old. See "Creating Customized Bringover/Putback File Lists" on page 40.
`backup/`	Stores information that Configuring uses to "undo" a Bringover or Putback transaction. See "Undoing Changes to a Workspace" on page 29.
`children`	Contains a list of the workspace's child workspaces. The names of child workspaces are entered into the workspace's `children` file during the Bringover Create transaction. Configuring consults this file to obtain the list of child workspaces. When you delete, move, or reparent a workspace, Configuring updates the `children` file in its parent.
`conflicts`	Contains a list of files in that workspace that are currently in conflict. See Chapter 6 for more information about conflicts and how to resolve them.
`description`	Contains a descriptive name and detailed description of a TeamWare workspace created by the user. See "Giving a Workspace a Descriptive Name" on page 43.
`Freezepoints/`	Created when you set up auto-freezepoints. See "Automatically Generating Freezepoints" on page 114.
`history`	An historical log of transactions and updated files that affect a workspace. See "Viewing Workspace History" on page 31 for more information.

TABLE 12-1 Contents of the `Codemgr_wsdata` Metadata Directory *(Continued)*

File/Dir Name	Description
`locks`	To assure consistency, Configuring locks workspaces during Bringover, Putback, and Undo transactions. Locks are recorded in the `locks` file in each workspace; Configuring checks that file before acting in a workspace. See "Removing Workspace Locks" on page 63.
`nametable`	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 `nametable` file is used by Configuring during bringover and putback to accelerate the processing of files that have been renamed. If this file is not available, Configuring rebuilds it automatically during the next Putback or Bringover transaction. See "Renaming, Moving, or Deleting Files" on page 94.
`notification`	Permits Configuring to detect events that involve that workspace and to send email in response to the event. See "Notifying Users of Transactions" on page 42.
`parent`	The `parent` file contains the path name of the workspace's parent workspace and is created by the Bringover Create transaction, or by the `reparent` command if the workspace was originally created with the `create workspace` command (and thus had no parent). Configuring consults this file to determine a workspace's parent. When you delete, move, or reparent a workspace, Configuring updates the `parent` file in its children.
`putback.cmt`	A cache of the text of the comment from the last *blocked* Putback transaction. Configuring caches the comment in `putback.cmt` so that you can retrieve the original text when you re-execute the transaction.

Configuring Defaults Files

When you change Configuring behavior using the Tool Properties window, click the Apply button to preserve the changes in runtime configuration files in your home directory. The runtime configuration files are consulted by Configuring when it starts; your changes are used as the default values.

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

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

You can also control which workspaces are loaded by default using environment variables (see "Configuring Environment Variables" on page 51). You can save a default list of files to bringover/putback (see "Generating a Customized List of Files" on page 41).

The `access_control` File

Chapter 4 introduces the concept of controlling who can perform which transactions on a workspace. When you set permissions in the Access Control tab (under Workspace ➤ Properties), TeamWare writes the permission in the `access_control` file. TABLE 12-2 shows the default contents of `access_control` after you create a workspace.

TABLE 12-2 Default Access Control Permissions

Operation	Default Permission
bringover-from	all users
bringover-to	creator
putback-from	all users
putback-to	all users
undo	all users
workspace-delete	creator
workspace-move	creator
workspace-reparent	creator
workspace-reparent-to	all users

You can manually edit the `access_control` file to control which users have access to a workspace. TABLE 12-3 shows all of the value types you can specify to control access to your workspaces and what the entries mean.

TABLE 12-3 Workspace Access Control Values

Value	Meaning
`@engineering`	All users in the net group named `engineering` can execute this operation.
`-@engineering`	No users from the net group named `engineering` can execute this operation. "`-`" denotes negation.
`@special -user2 @engineering`	All users in the net groups `special` and `engineering` can execute the operation; `user2` cannot (unless `user2` is in the `special` netgroup). "`-`" denotes negation.
`user1 user2`	The users `user1` and `user2` 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 can execute the operation.

How Configuring Merges Files

This discussion assumes that you are familiar with SCCS, including the concept of branching. Branching is defined in "How To Read a File's History: Deltas, Branches and Versions" on page 92. SCCS is described in detail in the Solaris *Programming Utilities* manual.

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

When a file is copied (using a 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 derived, and the file's entire delta and comment history are available.

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

Merging Files That Do Not Conflict

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

To accomplish the merger, the Configuring program determines where the delta histories diverge and adds (to the destination workspace) only the deltas that were created in the source workspace since they diverged. To determine where the histories diverge, the Configuring program compares the delta tables in both the parent and child history files; information used in this comparison includes comments and data, such as who created the delta and when.

FIGURE 12-1 contains an example of a Putback transaction where the Configuring program adds deltas 1.3 and 1.4 from the child workspace to the SCCS history file in the parent.

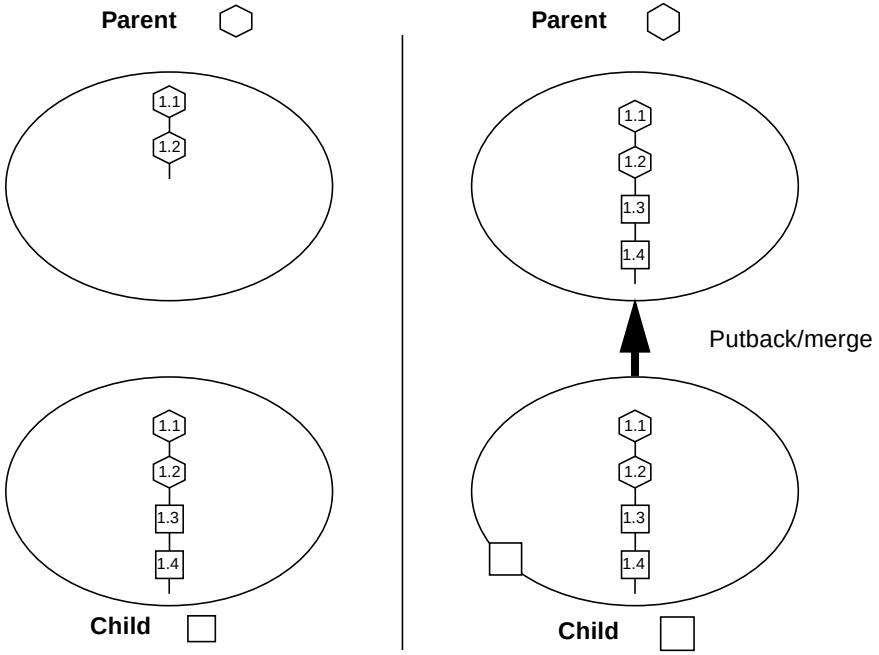


FIGURE 12-1 Updating a File in the Destination Workspace That Has Not Changed

Merging Files That Conflict

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

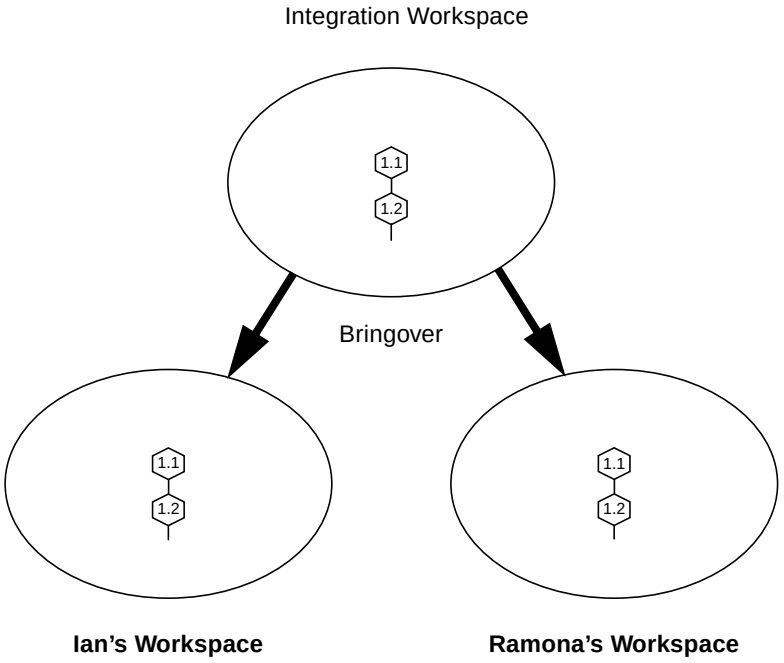
When file contents conflict, Configuring helps you to resolve the potentially conflicting changes that were made to the file and preserves the file's delta, administrative, and comment history. Configuring merges the SCCS deltas from the parent into the history file in the child. Configuring's Resolve transaction is then used to resolve the conflict in the child. For details on resolving conflicts, see "Resolving Differences" on page 77.

How Merging Tracks Deltas

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

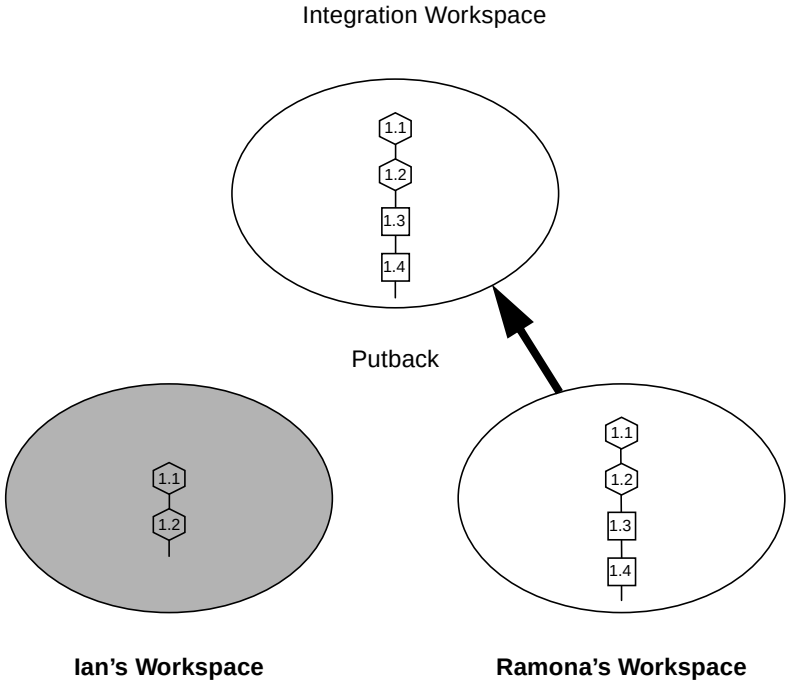
- The default delta (the point at which the next delta is added to the SCCS delta tree) is identified by an unattached descending line.
- You can use Versioning to graphically display SCCS delta trees in much the same way they are depicted here.

Both Ian and Ramona copy the same file from the integration workspace with the Bringover transaction. The file is new in both workspaces, so the SCCS history file is copied to both. Integration Workspace represents the parent workspace that both Ian and Ramona use.



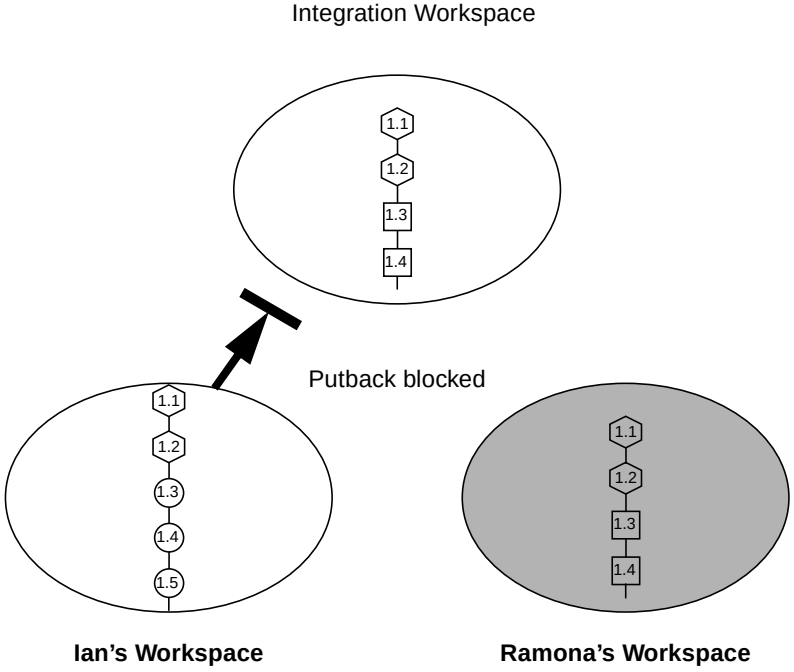
Ramona makes changes to the file, creating two new deltas: 1.3 and 1.4, and then puts the file back into the integration workspace (with the Putback transaction). Configuring appends the two new deltas to the parent SCCS delta tree.

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.



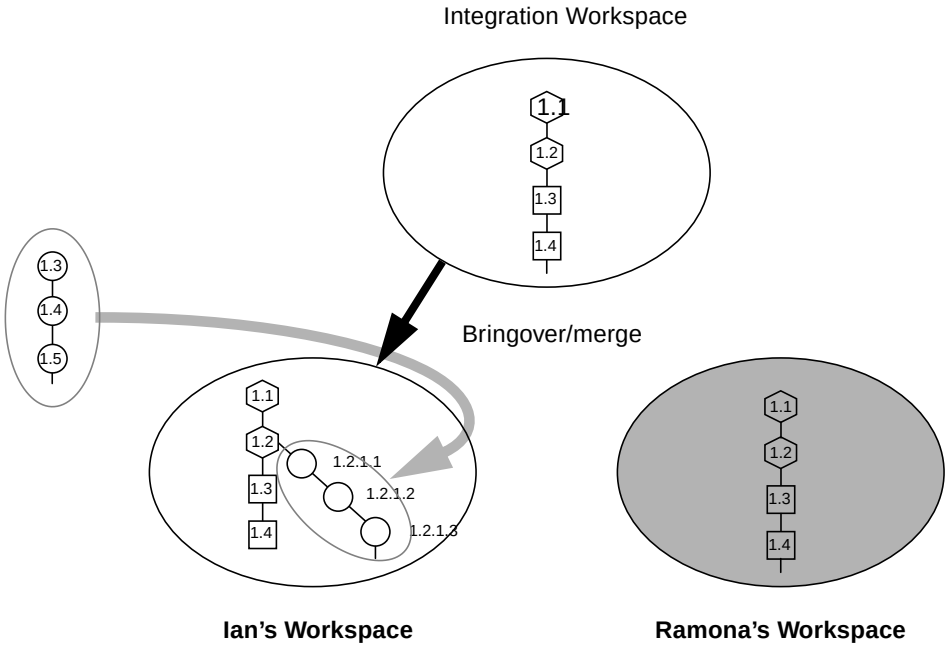
In the meantime, Ian also changes the file (creating three new deltas: 1.3, 1.4, and 1.5) and now attempts to put back the file into the integration workspace.

Configuring blocks the Ian's Putback transaction of because the files are in conflict. Ramona's changes that she put back would be overwritten. Ian must first incorporate the changes made by Ramona into his work.



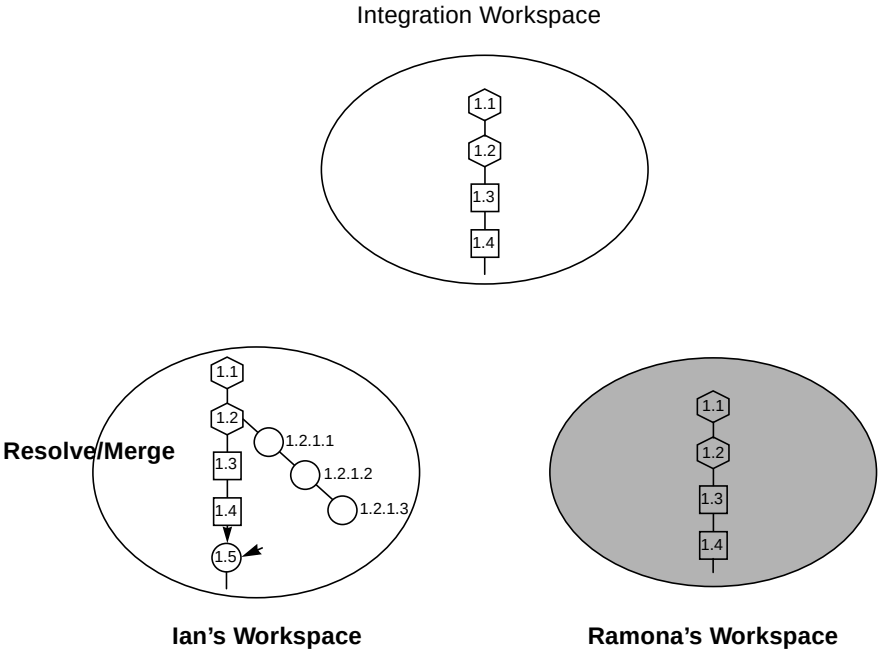
Ian brings over the file that now contains the changes made by Ramona into his workspace from the integration workspace. The deltas that Ramona created are added into the child SCCS history file by Configuring.

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 Ian 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).

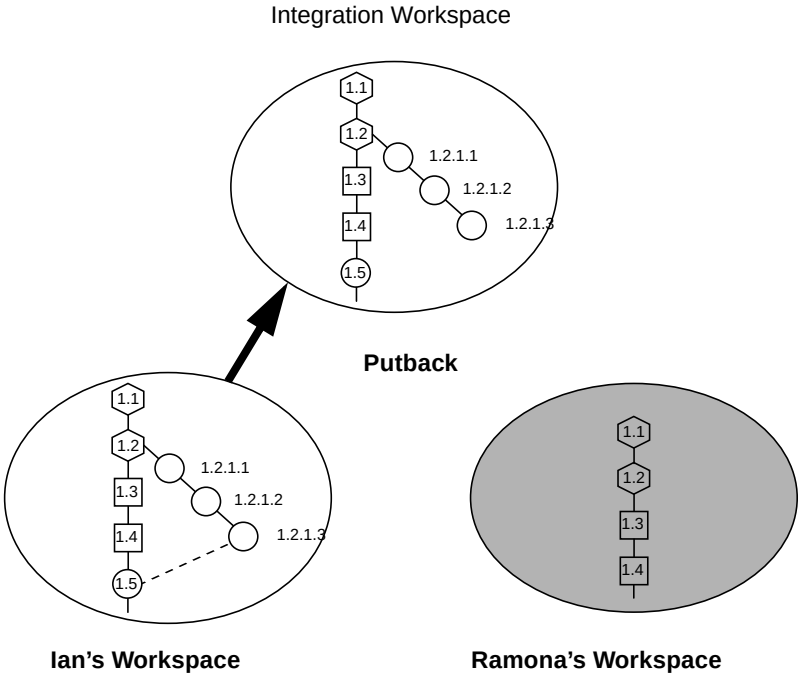


Ian resolves the conflict in his workspace using the Resolve transaction (see “Resolving Differences” on page 77 for details regarding conflict resolution). Ian uses the Resolve transaction to help him decide how to merge the versions of the file represented by SIDs 1.2.1.3 and 1.4. When he commits the changes, the Resolve transaction places the newly merged contents into a new delta 1.5:

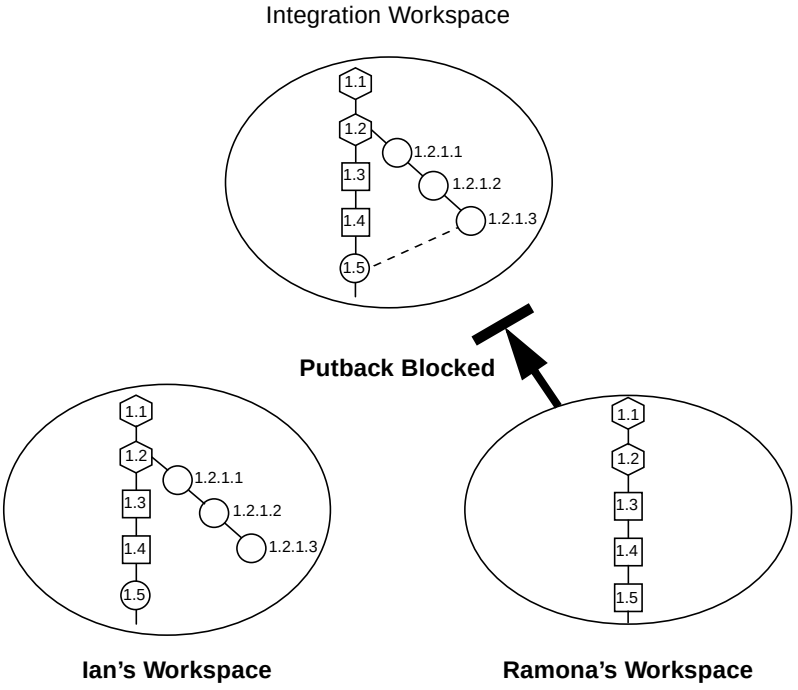
- The new delta, 1.5, is contained in a circle because Ian created it.
- The newly created delta is now the default location for any new work Ian creates.



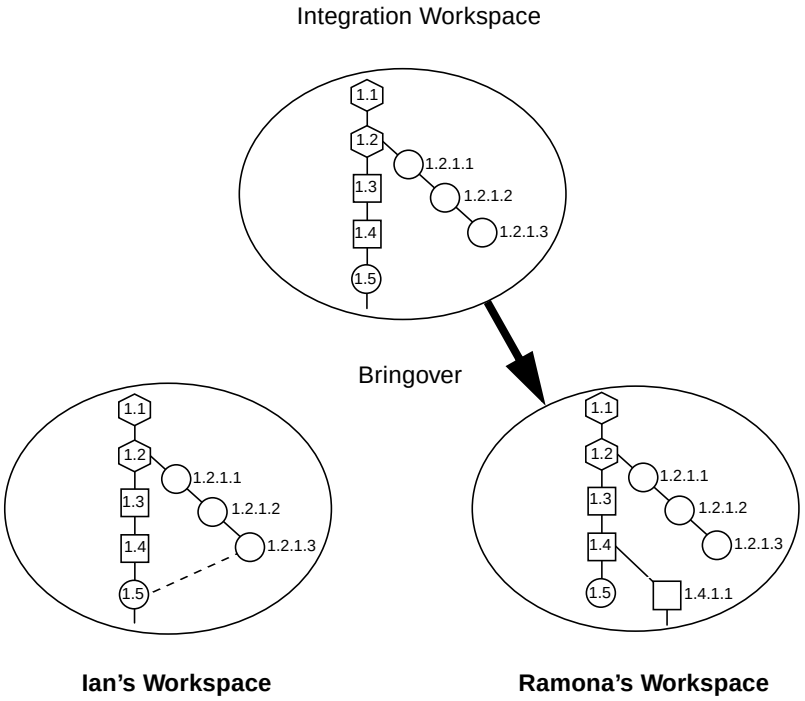
With the conflict resolved, Ian puts back the file into the integration workspace. The branch and the newly created delta are added to the SCCS history file in the integration workspace.



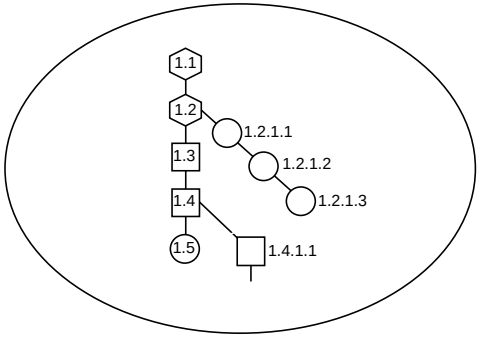
Ramona makes another change to the file in her workspace, creating delta 1.5. She attempts to put back the new work to the integration workspace, but the Putback transaction is blocked because it conflicts with the newly merged delta 1.5 that Ian had put back.



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



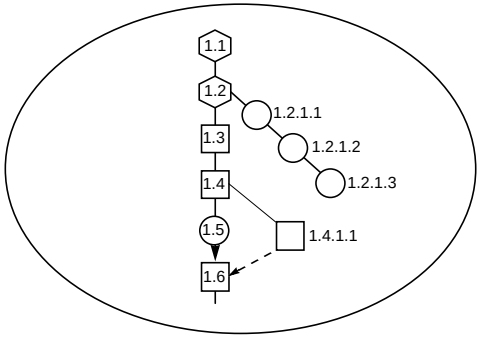
As in the previous case, Configuring appends the delta that Ramona created to the last common delta on the delta tree trunk as a branch and renumbers it appropriately. 1.5 becomes 1.4.1.1. 1.4.1.1 remains the default delta. Any new deltas created in the child before the conflict is resolved will be added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



Ramona's Workspace

Using the Resolve transaction, Ramona resolves the conflict merging the differences between 1.5 and 1.4.1.1 to create the new delta 1.6:

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



Ramona's Workspace

This merge example has shown what happens when you have 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. When they attempt to put their changes back, conflicts occur. Configuring manages deltas to ensure that no one's changes are overwritten and manipulates the SCCS history file so that the file history is accurate and complete.

About SCCS Mergeable IDs

This section explains why SCCS Mergeable IDs (SMIDs) are necessary, how to translate SCCS delta IDS (SIDs) to SMIDs, and how to translate SMIDs to SIDs.

The use of SMIDs ensures that every delta is uniquely identifiable, even if its SID is changed. A SMID is a number generated using the Xerox Secure Hash Function. When you use Freezepointing to create a freezepoint file, it calculates the SMID for both the current delta and the root delta in the SCCS history file. Using both of these values, Freezepointing can identify a delta in a file even if its SID has been changed.

Why SMIDs are Necessary

When Configuring encounters a file conflict during a Bringover Update transaction (file is changed in both the parent and child workspaces), it merges the new deltas from the parent workspace into the SCCS history file in the child. When this merge occurs, the deltas that were created in the child are moved to an SCCS branch off of the delta that both deltas have in common (common ancestor).

When Configuring relocates the child deltas to a branch, it changes their SID. If SIDS were used in freezepoint files to identify deltas, this relocation would invalidate the information contained in the freezepoint file. For that reason, SIDs cannot be used to identify deltas after conflicting SCCS histories have been merged.

SMID/SID Translation

You can translate SMIDs into SIDs and SIDs into SMIDs using the `freezept sid` and `freezept smid` commands. This is useful if you want 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 want to calculate a SMID.

- Use the `-a` option to calculate a SMID for all of the SIDS in *file*.
- For convenience, you can use the `-s` option to specify a directory from which *file* is relative.

Examples

```
example% freezept smid -r 1.38 module.c
SID 1.38 = SMID “f5b67794 705f0768 a89b1f4 588de104”
```

```
example% freezept smid -a bringover.1
SID 1.1 = SMID “b05b0a2f 1db5246e 1a466014 707e38f5”
SID 1.2 = SMID “d6a5c61f 5634f0ef 9847a080 d0d7b212”
SID 1.2 = SMID “e31acdd5 6c1232e2 9e81c287 1edb2f41”
SID 1.3 = SMID “c34c91b4 a818622a 2457356a 489b2728”
SID 1.4 = SMID “98c0fd8d 889563fb cf722c2b 6afc9636”
SID 1.5 = SMID “b1e24be3 752fec3e df2d2717 a9b3f1fa”
SID 1.6 = SMID “2b93d39 1ea2f6ba 9814320c bc609acb”
SID 1.7 = SMID “1db7d640 42b0f009 35c60d7b b230bd85”
SID 1.8 = SMID “906dfe9a ca7e2d6c a64da5be 4baef254”
```

Translating SMIDS to SIDS

Use the `freezept sid` command to translate SMIDs to SIDs. The syntax is:

`freezept sid` [**`-w`** *workspace*] [**`-m`** “*SMID*”] [**`-a`**] *file*

- Use the `-m` option to specify the SMID (in file *file*) for which you want 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 1edb2f41” = 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 2436ca1c” = SID 1.10
SMID “46de4131 b95b9973 93958a07 b960074c” = SID 1.11
```

APPENDIX **A**

Error and Warning Messages

This appendix describes the error and warning messages displayed by Sun WorkShop TeamWare.

The chapter contains the following sections:

- Error Messages
- Warning Messages

All messages are numbered and are listed in numerical order. For each message, the meaning of the message and a possible solution for the error are provided.

Error Messages

TABLE A-1 describes error messages, their meanings, and possible solutions.

TABLE A-1 Configuring Error Messages

1000 - 1999	System Errors Error messages between 1000 and 1999 report errors from operating system calls made by Configuring commands. They consist of a short Configuring message and an appended system error message and number. Refer to operating system documentation for information regarding these errors.
2000	`Line too long or unexpected end of file in` *file_name* Meaning: While reading the *file_name*, a line was encountered that contained too many characters for a Configuring command to buffer. The maximum line length is 1024 characters. Solution: Reduce the size of the long line and re-execute the command.

TABLE A-1 Configuring Error Messages *(Continued)*

2001	`Must specify a [child]* workspace either with the -w option or via the CODEMGR_WS environment variable` Meaning: The Configuring command could not determine the workspace on which to act. Configuring commands attempt to acquire the workspace path name in the following order: • 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 *When the error is reported by Bringover and Putback the word child is included, when reported by Undo and Resolve it is not included. Solution: Specify the workspace path name using one of the methods listed above.
2003	*directory_name* `is not a workspace` Meaning: The directory specified in the command is not a Configuring workspace. Configuring workspaces are distinguished by the presence of the `Codemgr_wsdata` directory in the top level directory. Solution: Specify a different workspace name or use the `workspace create` command or the GUI File ➤ Create Workspace command to convert the directory into a workspace.
2004	`Workspace` *workspace_name* `doesn't have a parent workspace` Meaning: A Configuring command (Bringover or Putback) could not complete execution because a parent workspace could not be found for workspace workspace_name. Solution: Use the `workspace parent` command or the GUI Edit ➤ Parent menu item 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. Solution: Mount the file system that contains the parent workspace and reissue the command.
2006	`Filename` *file_name* `has too many ".." path components in it` Meaning: Relative file names specified to Configuring commands are interpreted as being relative to the root directory of the workspace. If a file name contains `".."` components, it is possible for one of the `".."` components to reach a directory that is hierarchically above the workspace root. Solution: 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`

TABLE A-1 Configuring Error Messages *(Continued)*

	Solution: Check NIS server and maps.
2008	`No version number in file` *file_name* Meaning: When a Configuring 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. Solution: Check the integrity of *file_name*. The version string may have been removed when the file was edited. If the version string is missing, and the file is not otherwise corrupted, use the `workspace create` command or the GUI File ➤ Create Workspace menu item 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 Configuring command attempted to execute command_name and was unable to because the shell was killed by signal. Solution: Re-execute the Configuring command.
2010	`Command` *command_name* `failed, could not execute the shell, /bin/sh` Meaning: A Configuring command could not start a shell. This indicates that some system resource, such as swap space or memory, was insufficient. Solution: Check system resources.
2011	`Command` *command_name* `killed by signal` signal Meaning: A command started by a Configuring command received a signal. Solution: Re-execute the command. If the error reoccurs, refer to the Solaris documentation for information about the signal.
2012	`Command` *command_name* `exited with status` *status* Meaning: Configuring expects commands it executes to exit with a status of zero indicating successful completion. Configuring considers it an error if a command exits with a nonzero status. Solution: 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 file list program (FLP) FLP_name specified for the Bringover or Putback transaction could not be found in either the parent or child workspace Solution: Check the path name of the intended FLP and re-execute the transaction.

TABLE A-1 Configuring Error Messages *(Continued)*

2014	`Could not execute` *program_name* Meaning: A Configuring command attempted to execute another program and was unable to do so. Solution: 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. Solution: 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 Configuring command could not be found. Solution: Ensure that the path name was specified correctly.
2017	`Can't open file` *file_name* `so can't get comments for check in` Meaning: Configuring stored check in comments in a temporary file and was unable to open that file to read the comments. Solution: Check file permissions and other file system problems that would prohibit opening the file.
2018	`Can't reparent a workspace to itself` Meaning: An attempt was made (either as part of a transaction, or by using an explicit reparent command) to make a workspace its own new parent. Solution: Re-execute the command, specifying a different parent.
2019	`Internal error: unknown locktype` *lock* `in workspace` *workspace_name* Meaning: The workspace lock file (`Codemgr_wsdata/locks`) is corrupted. An unknown lock value was found. Solution: Edit the lock file to repair the damage. For more information, see the `locks(4)` man page or "Removing Workspace Locks" on page 63.
2020	`You must specify a workspace name` Meaning: The Configuring command could not determine the workspace on which to act. Configuring commands attempt to acquire the workspace path name in the following order: • 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 Solution: Specify the workspace path name using one of the methods listed above.

TABLE A-1 Configuring Error Messages *(Continued)*

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: To ensure consistency, Configuring interworkspace commands lock workspaces while they read and write data in them. The command you issued could not obtain a lock because the workspace is already locked. While Configuring is reading and examining files in the parent workspace during a Bringover transaction, it obtains a *read-lock* for that workspace. While it is manipulating files in the child workspace, it obtains a *write-lock*. Read-locks may be obtained concurrently by multiple Configuring commands that read files in the workspace. No commands may write to a workspace while any read-locks are in force. Only a single write-lock can be in force at any time; no Configuring command may write to a workspace while a write-lock is in force. Lock status is controlled by the `Codemgr_wsdata/locks` file in each workspace. Solution: 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 Options ➤ Workspace menu item and selecting Locks from the Category list box in the Workspace Properties dialog box, or use 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 nonexistent subcommand was specified. Solution: For the list of valid subcommands for each command, type the command and specify the help subcommand.
2023	`Integration Request failed` Meaning: The Putback validation is activated in the parent workspace and the validation program returned a non-zero exit status. Solution: Re-execute the `putback` command specifying the correct modification request id.
2024	`File` *file_name* `has no deltas` Meaning: The SCCS history file *file_name* contains no deltas, therefore, it cannot be processed. Solution: Perhaps the history file was 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 Configuring command attempted to execute another program and was not able to find it.

TABLE A-1 Configuring Error Messages *(Continued)*

	Solution: 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 Configuring 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. Refer to the Solaris SCCS documentation regarding SCCS history file format.
2027	`Corrupted file -` *file_name*`, line` *line_number* Meaning: A Configuring command was unable to read a workspace metadata file (a file in the Codemgr_wsdata directory). Illegal characters were found in line *line_number*. Solution: Check and repair the file. All Configuring metadata files are ASCII text files and can be edited. See "Workspace Metadata Directory" on page 157 for more information about the files in the Codemgr_wsdata directory or the *file_name*(4) man page.
2028	`Could not find the` *command_name* `command in PATH` *path_name* Meaning: A Configuring command attempted to execute another program and was not able to find it. Solution: 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. Solution: Use the `edit m` subcommand (edit the merged result) to resolve the conflicts and then save the file. Conflicts are marked with ^<<<<<<<.
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. Solution: Use the `list` subcommand to list the files and determine the correct number of the file you want to specify.
2031	`Can't find home directory so can't write to file` *file_name* Meaning: A Configuring command was unable to find the user's home directory and cannot locate file *file_name*. This usually indicates a problem with NIS maps. Solution: Check NIS server and appropriate NIS maps.
2032	`Can't parse line in file` *file_name*`:` *line*

TABLE A-1 Configuring Error Messages *(Continued)*

	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. Solution: Correct the file `~/.codemgr_resrc` file so that it includes only valid entries. For information regarding these entries, see the `resolve`(1) manual page.
2033	`Must specify a directory list either as arguments or via the CODEMGR_WSPATH environment variable` Meaning: This message is reported by the `workspace list` command when a directory (or list of directories) was not specified correctly. 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. Solution: 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 Ibuilt-in default` Meaning: A Configuring command attempted to verify access permission for a workspace operation (for example: bringover-from, putback-to, reparent-to). An internal consistency check failed. Solution: Contact your local service representative.
2035	`Access control file does not exist` Meaning: A Configuring 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. Solution: 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` command or the GUI File ➤ Create Workspace menu item and copy the file from the newly created workspace. For more information refer to Chapter 4 or the `access_control`(4) man page.
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. Solution: Proceed with the conflict resolution process without specifying the ancestor (`a`) as an argument to the `diff`, `edit`, and `more` subcommands.

TABLE A-1 Configuring Error Messages *(Continued)*

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). Solution: Specify one of the valid arguments: `a, c, p, m`. See the `resolve`(1) man page for more information.
2039	`File` *file_name* `is probably not an s-file on line` *line_number* `expected ^A, but got` *char* Meaning: A Configuring command expected file_name to be an SCCS history file. Based on the format, it is either not a history file or it was corrupted. Solution: Refer to Solaris SCCS documentation regarding SCCS history file format.
2040	`File` *file_name* `has not been merged. Use the merge subcommand first or use the filemerge subcommand` Meaning: An attempt was made to commit (save) a file that had not yet been merged. Solution: Merge the file using either Filemerge or the `twmerge` command. For more information about the `twmerge` command see the `twmerge`(1) man page.
2041	*path_name* `is not a workspace or a directory` Meaning: The string path_name specified in the Bringover Create transaction is not a Configuring workspace or a directory. Solution: Specify a different workspace or directory name.
2042	`Can't create ToolTalk message, error = TT_error_code` Meaning: The resolve command communicates with the Merging program via the ToolTalk service. ToolTalk is an interapplication communication service distributed with the Solaris OpenWindows windowing system. In this case the `resolve` command called a ToolTalk routine to create a ToolTalk message for Merging. The ToolTalk routine could not create the message and passed back TT_error_code. Solution: 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. Solution: See the Solaris SCCS documentation for more information.
2044	`Unable to create a temporary name from template` *temp_file_name* Meaning: A Configuring command was unable to create a temporary file for its use. This is a Configuring internal error.

TABLE A-1 Configuring Error Messages *(Continued)*

	Solution: 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*. Solution: 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 Configuring 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 Configuring commands read and write to these files, they check to determine whether the file contains the version that the command expects. In this case the command expected to find *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. Solution: Make sure that the most current versions of the Configuring binaries are being accessed.
2047	`Do not know how to convert file` *file_name* `from version` *found_version_number* `to` *current_version_number* Meaning: A command found a metadata file with a version number earlier than 1. Each Configuring 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 Configuring commands read and write to these files, they check to determine whether the file contains the version that the command expects. It is anticipated that when new versions of Configuring binaries and metadata files are released, the formats of some of these files may change. Commands contain code to make this conversion. Solution: The version string in the metadata file must have been inadvertently changed during editing. Check the file and make sure that the first line has the correct version number.
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, Configuring attempts to take the arguments from the workspace's `Codemgr_wsdata/args` file. Solution: Re-enter the command and ensure that you've included the correct number of arguments.
2049	`Could not determine where` *file_name* `is mounted from`

TABLE A-1 Configuring Error Messages *(Continued)*

	Meaning: Configuring 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 contained in `/etc/mnttab` is no longer present. Solution: Remount the file system that contains *`file_name`*.
2050	`Could not determine the absolute pathname for` *file_name* Meaning: A Configuring command was unable to read a directory. This indicates some corruption in the file system; for example, incorrect directory permissions. Solution: 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. Solution: Change the name in one of the directories.
2052	`Corrupted file -` *file_name*`, text after BEGIN,` *line number*`. Can't send notification` Meaning: A Configuring command encountered an error when reading the workspace notification file `Codemgr_wsdata/notification`. The `BEGIN` statement that delimits the list of files/directories for which notification is requested must be the only text on the line, other text was encountered. The Configuring command cannot correctly parse the request; if the file contains a notification request, it cannot be sent. Solution: Edit the notification file and enter the appropriate `BEGIN` statement. See the `notification`(4) man page or Chapter 4 for more information on its format.
2053	`Corrupted file -` *file_name*`, text after END,` *line number*`. Can't send notification` Meaning: A Configuring command encountered an error when reading the workspace notification file `Codemgr_wsdata/notification`. The `END` statement that delimits the list of files/directories for which notification is requested must be the only text on the line, other text was encountered. The Configuring command cannot correctly parse the request; if the file contains a notification request, it cannot be sent. Solution: Edit the notification file and enter the appropriate `END` statement. See the `notification`(4) manual page or "Notifying Users of Transactions" on page 42 for more information on its format.

TABLE A-1 Configuring Error Messages *(Continued)*

2054	`Corrupted file -` *file_name*`, missing END,` *line number*`. Can't send notification` Meaning: A Configuring command encountered an error when reading the workspace notification file `Codemgr_wsdata/notification`. The `END` statement that delimits the list of files/directories for which notification is requested, is missing. The Configuring command cannot correctly parse the request; if the file contains a notification request, it cannot be sent. Solution: Edit the notification file and enter the appropriate `END` statement. See the `notification`(4) man page or "Notifying Users of Transactions" on page 42 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 was corrupted. Solution: Fix the file, or copy in a new version.
2056	`Badly formatted line in` *file_name*`:` *line_number* Meaning: A Configuring command was reading a temporary log file left over from an aborted Bringover or Putback operation and encountered a malformed line. This indicates that the file was corrupted. Solution: 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. Solution: 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. Solution: 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. Solution: 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. Solution: Reorder the delta numbers, or copy in a new version of the file using the Bringover or Putback transaction.

TABLE A-1 Configuring Error Messages *(Continued)*

2061	`Must have DISPLAY environment variable set to invoke twmerge or filemerge` Meaning: Merging is an XWindows program. The `DISPLAY` environment variable is used to determine where the program displays its windows. If you are running CDE or OpenWindows, this variable is usually set for you. For other XWindow display servers, you may need to set this variable manually before starting Merging. Solution: Determine the correct setting for your display server and set the `DISPLAY` environment variable appropriately.
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. Solution: Reconcile the file permissions (for example, check out the file 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. Solution: 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: Configuring commands will not delete directories or files that are symbolic links. You must delete the physical copy of the file. The appropriate command line is provided. Solution: Use the `workspace delete` command to delete *workspace_name*.

TABLE A-1 Configuring Error Messages *(Continued)*

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. Solution: 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 ➤ Properties menu item and selecting the Access Control tab in the Workspace Properties dialog box, or by editing the `access_control` file directly. For more information, see Chapter 4 or the `access_control`(4) man page.
2066	`Corrupted file -` *file_name*`, whitespace in` pathname`, line` *line_number*`. Can't send notification` Meaning: A Configuring command encountered an error when reading the workspace notification file `Codemgr_wsdata/notification`. A white space character was encountered in a line where a single path name was expected. Solution: Edit the `Codemgr_wsdata/notification` file to remove the whitespace characters from the line. See the `notification`(4) man page or "Notifying Users of Transactions" on page 42.

TABLE A-1 Configuring Error Messages *(Continued)*

2067	`Corrupted file -` *file_name*`, missing notification event, line` *line_number*`. Can't send notification` Meaning: A Configuring command encountered an error when reading the workspace notification file `Codemgr_wsdata/notification`. The Configuring event (for example, bringover-to) was not specified. Solution: Edit the `Codemgr_wsdata/notification` file to add the correct event.
2068	`SCCS error for file` *file*`-can not create lock file(cm4)` Meaning: Someone else is updating the SCCS file or the p-file, or you do not have write permission for the directory in which the SCCS file resides. Solution: If someone else is updating the SCCS file or the p-file, wait until they release the lock and then try again to access the file. If you do not have write permission in the directory, you cannot create a lock file in that directory until you obtain write permission.
2069	Not used

2070	`Description file is corrupted: has no name`
	`Description file is corrupted: has no keyword for description`
	
	Meaning: The workspace description metadata file (`Codemgr_wsdata/description`) is corrupted.
	
	Solution: Recreate workspace description metadata file.

2071	`You must specify either -m or -c option`
	`Only one of -m or -c option is allowed`
	
	Meaning: The `workspace find` command accepts either `-m` or `-c` options. These options cannot be specified together and one of these options should always be specified
	
	Solution: Re-execute the `workspace find` command and ensure that you have specified the correct arguments.

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.
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. Solution: Check the lock numbers for the workspace using the `workspace locks` command and enter a valid number.

TABLE A-1 Configuring Error Messages *(Continued)*

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. Solution: 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. Solution: 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 Configuring command. Solution: 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 text in error is displayed. Solution: 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: Configuring attempted to clear the backup area directory_name so that it could backup a new transaction. Configuring was not able to delete or rename the directory out of the way. The most likely cause is that file permissions have been changed for the directory. Solution: Check directory permissions for *directory_name*. Default Configuring permissions for this directory are 777.
2081-2087	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. Configuring uses the root delta to distinguish between files. The `workspace updatenames` command enables Configuring to distinguish between the files.

TABLE A-1 Configuring Error Messages *(Continued)*

	Solution: Execute the `workspace updatenames` command and then re-execute 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: Configuring commands will not move directories or files that are symbolic links.
	
	Solution: Move the workspace to a name that is not a symbolic link.

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. Configuring uses the root delta to distinguish between files. The `workspace updatenames` command enables Configuring to distinguish between the files.
	
	Solution: Execute the `workspace updatenames` command and then re-execute the command that spawned the error.

2091	`Internal error: hash table missing entry` Meaning: Internal error. Solution: 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. Configuring is unable to determine whether the file 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. Configuring can then correctly propagate the changes throughout the workspace hierarchy.
	
	Solution: Execute the `workspace updatenames` command and then re-execute the command that spawned the error.

2093	`Internal error: SmIDs not equivalent.` Meaning: Internal error.

TABLE A-1 Configuring Error Messages *(Continued)*

	Solution: Contact your local service representative.
2094	`Internal error: SmID not found` Meaning: Internal error. Solution: Contact your local service representative.
2095	`Could not make SmID unique in file` *file_name.* Meaning: Configuring command found files with identical root deltas. Subsequent attempt to differentiate these files failed. Solution: Execute the `workspace updatenames` command in the workspace that contains *file_name* and re-execute the command that spawned the error.
2096	Not used.
2097	`Cannot remove workspace` *workspace_name.* Meaning: The `workspace move` command was unable to delete the original workspace after a copy was done. Solution: Check permissions and manually remove the original workspace.
2098	`Cannot delete workspace` *workspace_name*`. Please check permissions.` Meaning: The `workspace delete` command is unable to remove specified workspace. Solution: Check permissions and then re-execute the command.
2099 - 2499	Not used
2500 - 2536	Internal errors Meaning: Error numbers 2500 through 2536 are Configuring program's internal errors. These errors indicate problems that users cannot correct. Contact your local service representative.
2537	`File` *parent_file* `has incompatible type with file` *child_file* `(text/binary).` Meaning: `Bringover/Putback` command detected that the *parent_file* type (text/binary) is not compatible with the type of the *child_file*.. Solution: Fix the file or copy in a new version of the file using Bringover or Putback translation.
2538	`Can't set autofreezepoint properties.` Meaning: Internal error. Solution: Call your local service representative.
2539-2600	Internal errors

TABLE A-1 Configuring Error Messages *(Continued)*

	Meaning: Error numbers 2539 through 2600 are Configuring program's internal errors. These errors indicate problems that users cannot correct. Contact your local service representative.
2610	`Only putback to` *workspace_name* `is currently allowed` Meaning: An attempt was made to bring over files to a workspace *workspace_name* which has putback validation activated. Only `putback` transactions are allowed to workspaces that have putback validation activated. Solution: Disable putback validation for the *workspace_name* and then re-execute the `bringover` command
5005	`Destination directory` *directory* `is not empty.` Meaning: Directory *directory* specified for `freezept extract` command is not empty. Solution: Re-execute the `freezept extract` command and specify a destination directory that is empty.

5009	`Invalid date format:`
	date
	`expected:`
	YYYY/MM/DD `or` *YYYY/MM/DD/hh/mm/ss*
	
	Meaning: The *date* specified for the `freezept` command has an incorrect format.
	
	Solution: Re-execute the freezept command specifying the date in a correct format, either *YYYY/MM/DD* or *YYYY/MM/DD/hh/mm/ss*.

5010	`The delta specified in freezepoint` *freezepoint_file* `for file` *file_name* `does not exist in workspace` *workspace_name.* Meaning: The `freezept extract` command found a reference in *freezepoint_file* to a delta that does not exist in file *file_name* in workspace *workspace_name*. The most probable reason for this error is that the freezepoint *freezepoint_file* was created from a workspace that was `freezept updated` in relation to the same workspace *workspace_name* that was specified for this `freezept extract` command. Solution: Re-execute the `freezept extract` command specifying the correct workspace.
5011	`Your directory contains SCCS files with duplicate smids.` Meaning: The directory specified for `freezept create` command contains files with identical root deltas. Solution: Create a new Forte TeamWare workspace from your directory and re-execute the `freezept create` command.
5103	`Freezepoint file` *freezepoint_file* `is corrupted.` Meaning: The specified freezepoint file is corrupted.

TABLE A-1 Configuring Error Messages *(Continued)*

	Solution: Fix the file, or copy in a new version and re-execute the *freezept* command.
5107	`Must specify a workspace with the -w option since there is no default workspace in freezepoint file` *freezepoint_file* Meaning: The specified freezepoint file does not contain a default workspace that should be used for `freezept update` or `freezept extract` commands. Solution: Re-execute the `freezept` command specifying the correct workspace name.
5108	`The number of errors during extract is` *number*`.` Meaning: The *freezept extract* command displays the number of encountered errors at the end of work. Solution: Not applicable.
5109	`There is no default workspace in freezepoint file` *freezepoint_file.* Meaning: The specified freezepoint file does not contain a default workspace that should be used for `freezept update` or `freezept extract` commands Solution: Re-execute the `freezept` command specifying the correct workspace name.
6000	`When the` *-l* `option is used, both the leftfile and rightfile arguments must be names of directories`. Meaning: In the case when *-l* option is specified for the `filemerge` command, the values specified for the leftfile and rightfile arguments in the commandline should be names of directories. Solution: Re-execute the filemerge command and specify the correct values for the leftfile and rightfile arguments.
6003	`Merging: Could not parse diff output line:` "*line*" Meaning: Internal error. Solution: Contact your local service representative.

Warning Messages

TABLE A-2 describes warning messages, their meanings, and possible solutions.

TABLE A-2 Configuring Warning Messages

2020	`Parent defined with the option -p :` *workspace* `differs from native parent :` *parent_workspace* Meaning: The workspace *workspace* specified with the `-p` option for the `bringover` or `putback` command is not a parent workspace of the child specified for the `bringover` or `putback` transaction. The actual parent of the child workspace is a *parent_workspace*. Solution: Not aplicable.
2023	`Removing a command's lock while the command is running may corrupt SCCS files in either the parent or the child workspace.` Meaning: Self-explanatory. Issued by `workspace locks` command. Solution: If you are sure that the command that set the lock you are going to remove has already terminated, then click the OK button. Removing a running command's lock may cause a corruption of the history files.
2601	`Could not remove backup directory old_dir_name, so it was renamed to` *new_dir_name.* Meaning: Configuring attempted to clear the backup area *old_dir_name* so that it could backup a new transaction. Configuring was not able to clear the backup directory by deleting it, but it was able to rename it out of the way to the name *new_dir_name*. The most likely cause is that file permissions were changed for the directory. Solution: Check directory permissions for *old_dir_name*. Default Configuring 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: Configuring could not find an SCCS history file in either workspace for *file_name*. Solution: 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 (““).

TABLE A-2 Configuring Warning Messages *(Continued)*

	Solution: 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. Configuring commands do not accept file names that contain whitespace characters. Solution: 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. Solution: 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. Solution: Advisory only; the command corrects the discrepancy. However, this could indicate that the parent's `children` file was corrupted.
2609	Not used
2610	*directory_name* `is not a workspace` Meaning: The directory specified in the command is not a Configuring workspace. Configuring workspaces are distinguished by the presence of the `Codemgr_wsdata` directory in the top level directory. Solution: Specify a different workspace name or use the CLI `workspace create` command or GUI File ➤ Create Workspace menu item 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. Solution: Check to be sure the name was specified correctly.
2612	`Workspace` *workspace_name* `does not have Codemgr_wsdata/args file` `Workspace` *workspace_name* `has empty Codemgr_wsdata/args file.`

TABLE A-2 Configuring Warning Messages *(Continued)*

	Meaning: The workspace specified for the *bringover* or *putback* command does not have args metadata file or has an empty args metadata file. In this case, the *bringover*/*putback* command cannot determine default files/directories for the transaction. Solution: Re-execute the *bringover* or *putback* command and specify, on the command line, the files/directories to be used for the transaction
2613	`Filename` *file_name* `has too many ".." path components in it - ignored` Meaning: Possible causes include that the Configuring command cannot resolve the path name into a: • workspace-relative file name • fully qualified workspace name Solution: 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 Configuring command to buffer. The maximum line length is 1024 characters. This indicates that `nametable` has been corrupted. Solution: Configuring automatically rebuilds the nametable. 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. Solution: Configuring automatically rebuilds the nametable. This takes some time.
2616	Not used
2617	`Unexpected name table editlog record type` *type_number* `- ignored` Meaning: A Configuring command was reading a temporary log file left over from an aborted Bringover or Putback operation and encountered a malformed record. This indicates that the file has been corrupted. Solution: 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 Configuring notification facility failed to open the file *file_name*. As a result, notification mail is not sent for the current operation. Solution: Check file permissions for *file_name*.
2619	Not used
2620	`Can't fork process to send notification`

TABLE A-2 Configuring Warning Messages *(Continued)*

	Meaning: Lack of system resources (memory, swap space) prevented the Configuring notification facility from sending notification mail. Solution: 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. Configuring reserves this character to denote comments. Solution: 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 Configuring command was unable to remove locks in *workspace_name*. This may indicate that there is insufficient disk space, or that permissions on the file `Codemgr_wsdata/locks` were changed since the lock was originally written. Solution: Remove the locks by using the GUI Options ➤ Workspace menu item and selecting Locks in Category list box in the Workspace Properties dialog box, or by using 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 were not brought over as part of the Bringover transaction. Solution: Not applicable.
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. Solution: Not applicable.
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*. Solution: 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.`

TABLE A-2 Configuring Warning Messages *(Continued)*

	Meaning: Before beginning Bringover and Putback transactions, Configuring checks to determine whether the destination workspace root (top-level) directory is accessible for writing. This is not treated as an error condition because lower level directories within the workspace could be mounted from different areas and they may be accessible for writing. This warning is issued as an early warning that directory permissions might be set incorrectly. Solution: If write access is not intentionally denied, change the root directory permissions.
2628	`Not updating clear files because the 'check out' 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. Solution: If you want g-files to be updated as part of transactions, include the `get` command in your search path.
2630	`This workspace is being created over an existing directory` Meaning: You are converting an already existing directory into a Configuring workspace. Creating a workspace from an existing directory hierarchy consists of creating the `Codemgr_wsdata` metadata directory in the top-level directory. Once the directory becomes a workspace, its contents can be deleted using the Configuring `workspace delete` command. Solution: Not applicable.
2631	`File` *file_name* `not brought over because it is checked out and not writable in workspace` *workspace_name* Meaning: The file *file_name* was not brought over as part of the Bringover transaction because it is checked out (p-file exists) and writable in the child workspace *workspace_name*. The unusual state of this file indicates that it is safer not to process the file. Solution: 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. Solution: Correct the rename problem (see the rename error text) and re-execute the Configuring transaction.
2633	`Pathname "path_name" contains a symlink - ignored` Meaning: A symbolic link was encountered during a transaction. The directory is ignored and the transaction continues.

TABLE A-2 Configuring Warning Messages *(Continued)*

	Solution: Do not use symbolic links inside of Sun WorkShop Teamware workspaces.
2699	`File` *`file_name`* `ignored because it is renamed to` *`file_name1`* `in` *`source_workspace`* `and both` *`file_name`* `and` *`file_name1`* `exist in` *`destination_workspace`*`.` Meaning: A cyclic file rename has occurred. For example, there are two files, *`file_name`* and *`file_name1`*, in both parent and child workspaces. Then, in parent workspace, file *`file_name`* was renamed to *`file_name1`* and *`file_name1`* was renamed to *`file_name`*: `% workspace filemv` *`file_name tmp_file`* `% workspace filemv` *`file_name1 file_name`* `% workspace filemv` *`tmp_file file_name1`* The `bringover` cannot perform such rename (using temporary filename) in child workspace and issues this warning message. Solution: Perform corresponding rename sequence in the *`destination_workspace`*. For example: `% cd` *`destination_workspace`* `% workspace filemv` *`file_name tmp_file`* `% workspace filemv` *`file_name1 file_name`* `% workspace filemv` *`tmp_file file_name1`*

APPENDIX **B**

Troubleshooting

This appendix describes common problems with Sun WorkShop TeamWare and their solutions.

- Bringover and Putback Errors
- Environment Variables
- Process Monitoring
- SCCS Commands to Avoid
- SCCS History Files
- SCCS Errors
- Text Formatting Issues
- Version Verification
- Workspaces

Bringover and Putback Errors

TABLE B-1 shows some common bringover and putback errors.

TABLE B-1 Bringover and Putback Errors

Problem	Solution
Errors received while attempting a putback operation.	1. Preview the putback operation: find out how many file updates are to be made, without actually making them. From the Sun WorkShop TeamWare GUI, select the Preview checkbox in the Putback dialog. Or at the command line: `$ cd` *child_directory* `$ putback -n` (where *child_directory* is the top-level directory of your child workspace). 2. Verify there is sufficient disk space to write the putback operation's changes. Check available space on the disk where the parent workspace resides (see next solution). Also check the `/tmp` directory, and available swap space. 3. If still experiencing difficulties, try to do a partial putback in order to isolate the problem. A putback may consist of just a single directory or even a single file.
Receipt of the following error message while doing a putback: `putback: Line` *XXX* `has bad format...(Warning 2615)` (where *XXX* is a line number)	There may be insufficient disk space to write to the parent workspace. You can check available disk space by executing the `df -k` command in the directory where the parent workspace resides: `$ df -k` *parent_workspace_directory* Note: available disk space is shown in kilobytes.
Errors involving file names which contain spaces.	If using the command line interface, place any file names which contain spaces within double quotes. The Sun WorkShop TeamWare GUI will correctly handle transactions involving file names which contain spaces.

Environment Variables

TABLE B-2 lists some problems relating to environment variables.

TABLE B-2 Environment Variables and Related Problems

Problem	Solution
Any of the following error messages and symptoms during bringover, putback, workspace, and SCCS operations: • `cd : lstat /X failed` • A putback completes without error, but none of the updates are actually putback. • Executing the command, `sccs edit` *filename*, returns the prompt without doing anything.	Check your PATH environment variable and verify that `TeamWare/bin` is listed before any other applications in the `/bin` directory.

Process Monitoring

TABLE B-3 shows some common issues around process monitoring

TABLE B-3 Process Monitoring

Problem	Solution
How to monitor currently running Sun WorkShop TeamWare processes.	The following solution applies only to the Solaris platform. At the command line, type: `truss -o /tmp/truss_output -f \` `-vall -p` *TeamWare_PID* where *TeamWare_PID* is the process identification number of the currently running Sun WorkShop TeamWare process.

SCCS Commands to Avoid

The following SCCS commands should be avoided as their use may make alterations to and prevent Sun WorkShop TeamWare from being able to track SCCS history files:

- `cdc`
- `comb`
- `fix`
- `rmdel`

These commands also have undesirable side effects when used on files that exist in multiple workspaces that may eventually be merged.

A description of specific problems which may arise from these commands is shown in TABLE B-4.

TABLE B-4 SCCS Commands to Avoid

SCCS Command	Associated Problem
`cdc`	May cause unnecessary branching and confusing histories.
`comb`	Completely rebuilds the SCCS history file.
`fix`	A front-end for `rmdel`.
`rmdel`	SCCS restricts the use of the `rmdel` command to remove only the most recent (leaf) delta on a branch. If you remove a delta that also exists in another workspace, it is possible that another user will add a delta in the other workspace. In this case, the delta that was removed in your workspace will no longer be a leaf delta when the files are merged.

SCCS History Files

TABLE B-5 shows some common problems with SCCS history files

TABLE B-5 SCCS History Files

Problem	Solution
How to validate the SCCS history file.	Use the `sccs-val` command to do a diagnostic test on a specified file or files. The `sccs-val` command checks each file for: • history file corruption • history file read problems
How to restore a corrupted SCCS history file.	The SCCS history file contains a checksum. If the SCCS history file becomes corrupted, the checksum will need to be corrected. This can be accomplished with the following command: `sccs admin -z peogram.c` **Note**: If the SCCS history file is corrupted, the problem may be more serious than an incorrect checksum. Always backup your current changes before attempting to correct the SCCS history file.

SCCS Errors

TABLE B-6 describes some problems related to SCCS errors.

TABLE B-6 Problems with SCCS Errors

Problem	Solution

Either of the following error messages may result from Y2K compliance issues and older versions of SCCS:
• `ERROR [SCCS/s.`*foo*`.java] format error at line` *X* `(c04)`
• `bringover: Command get -s -G`*foo* `SCCS/s.`*foo* `exited with status 1 (Error 2012)`
(where *foo* is a real file name and *X* is a line number).
	Determine whether your version of SCCS is Y2K compliant by executing the following commands:
	`$ which get`
	to which the system should respond:
	`/usr/ccs/bin/get`
	then type:
	`/usr/ccs/bin/what /usr/ccs/bin/get`
	to which the system should respond with either:
	• 01/01/1997 - for Solaris 2.6
	• 04/23/1998 - for Solaris 7
	Any version of Solaris less than 2.6 is not Y2K compliant. If you find that your version of SCCS is not Y2K compliant, consult SunSolve Online for the appropriate patch.

Executing the command, `sccs diffs SCCS`, produces the error: `diff: SCCS/xxx0.0: No such file or directory`	The version of SCCS that was shipped with Solaris 2.6 has a known bug. Specifically, the `diffs` subcommand does not understand the `SCCS` directory as a parameter. To solve this problem you should obtain the appropriate patches for Solaris 2.6.

Text Formatting Issues

TABLE B-7 describes some issues related to text file formatting.

TABLE B-7 Issues With Text Formatting

Problem	Solution
Sun WorkShop TeamWare mistakes a text file for a binary file, producing an error such as: `bringover: Line too long or unexpected end of file in "//c/builds/`*child_workspace*`/`*filename*`" (Error 2000)`	Sun WorkShop TeamWare makes the assumption that a text file must end with a terminating newline, otherwise it is treated as a binary file. To prevent this problem ensure that all text files end in a terminating newline, adding them as necessary to your source files.

Version Verification

TABLE B-8 shows some common issues with version verification.

TABLE B-8 Version Verification

Problem	Solution
How to determine what version of Sun WorkShop TeamWare your are using.	From the Sun WorkShop TeamWare GUI, choose Help ➤ About TeamWare. Or alternatively, at the command line, type: `teamware -V`

Workspaces

TABLE B-9 describes some common issues with workspaces.

TABLE B-9 Workspace Issues

Problem	Solution
Concerns about workspace corruption.	1. The `workspace check` command audits workspaces and reports on inconsistencies. It checks files, access modes, parent-child relationships, and the condition of the history file. To execute the `workspace check` command, type: `workspace check` *workspace_name* (where *workspace_name* is the name of a workspace). The command exits with either of the following values: 0 = workspace is okay 1 = error 2. The workspace Auto-Freezepoint feature automatically snapshots a workspace before and after various transactions. While there is some overhead involved in using this feature, it is intended for master integration workspaces where integrity is of paramount importance. To more information about Auto-Freezepointing see page 114.
Incompatibility concerns regarding workspaces created using earlier versions of Sun WorkShop TeamWare.	There are no incompatibility issues regarding workspaces created using earlier versions of Sun WorkShop TeamWare.

Glossary

Access control A feature in Configuring you can use to control access to workspaces.

Branch (SCCS) A delta or series of deltas that are placed off of the main line of deltas in an SCCS history file.

Bringover Create The transaction used to copy groups of files from a parent workspace to a nonexistent child workspace. The new child workspace is created as a result of the transaction. All Configuring transactions are performed from the perspective of the child workspace; hence, the Bringover Create transaction "brings over" files to the child from the parent workspace. See also *Bringover Update*, *Workspace*, and *Putback*.

Bringover Update The transaction used to update an existing child workspace with respect to files contained in its parent workspace. All Configuring transactions are performed from the perspective of the child workspace; hence, the Bringover Update transaction "brings over" files to the child from the parent workspace. See also *Bringover Create*, *Workspace*, and *Putback*.

Child workspace A workspace that has a parent workspace listed in its `Codemgr_wsdata/parents` file. Development work is typically done in child workspaces and put back to parent workspaces after it has been tested. Configuring transactions are viewed from the child workspace perspective and all conflicts are resolved in the child workspace.

`Codemgr_wsdata` directory Every TeamWare workspace contains a "metadata" directory in its root directory named `Codemgr_wsdata`. Configuring stores data about the workspace in `Codemgr_wsdata`. The presence of this directory is the sole factor that defines it as a TeamWare workspace (as opposed to a normal directory). Configuring commands use the presence or absence of this directory to determine whether a directory is a workspace. See "Workspace Metadata Directory" on page 157 for more information.

Configuring The tool you use to manage Sun WorkShop TeamWare workspaces. See Chapter 2.

Conflict The condition that exists when a file has changed in both the child and parent workspace. Conflicts are identified by the Bringover Update transaction and are resolved by using the Resolve transaction.

Copy-Modify-Merge The concurrent development model upon which Configuring is based. Using this model, multiple developers concurrently *copy* sources from a common area, *modify* the source in isolation, and then *merge* those changes with changes made by other developers.

Create Used in Configuring transaction output. Files are said to be created if they exist in the source workspace and not in the destination workspace, and are copied into the destination workspace as part of a Bringover or Putback transaction.

Default path The branch in an SCCS history file upon which the next delta will be added. This is indicated in a history file by a solid line.

`def.dir.flp` The default FLP shipped with Configuring is `def.dir.flp`; this FLP recursively descends directory hierarchies and lists all files for which SCCS history files exist. See *FLP*.

Delta The set of differences between two versions of a file checked into SCCS. When you check in a file, SCCS records only the line-by-line differences between the text you check in and the previous version of the file. This set of differences is known as a delta. The file version that you initially checked out was constructed from a set of accumulated deltas. The terms delta and version are often used synonymously; however, their meanings are not the same. It is possible to retrieve a version that omits selected deltas. See *Version*.

FLP An FLP or *File List Program* is a program or script that generates a list of files that Configuring uses during Bringover and Putback transactions. See *def.dir.flp*.

Freezepointing The Sun WorkShop TeamWare tool you use to make snapshots of workspaces (or portions of them) at important junctures or "freezepoints." See Chapter 8.

g-file (SCCS) The working copy of a file retrieved from an SCCS history file by the `sccs-get` command.

History files When you initially put a file under SCCS control, a history file is created for the new SCCS file. The initial version of the history file uses the complete text of the source file. The initial history file is the file that further deltas are compared to. Owing to its prefix (s.), the history file is often referred to as the s.file (s-dot-file).

Integration Request Identifier A "password" required by Putback Validation before Configuring will allow a putback.

Integration workspace A workspace to which multiple developers put back their work.

Lock To assure consistency, the Configuring file transfer transactions Bringover and Putback lock workspaces while they are working in them. Locks are recorded in the `Codemgr_wsdata/lock` file in each workspace; the Configuring commands consult that file before acting in a workspace.

Merge To produce a single version of a file from two conflicting files. Accomplished with the assistance of the Merging tool.

Merging The Sun WorkShop TeamWare tool you use to merge two different versions of the same file. See Chapter 6.

MR Modification Request. A "password" required by Versioning before a file can be checked in.

Notification A Configuring feature that mails notices of events, such as changes to files or directories, to users.

Parent workspace A workspace that has a child workspace. Parent workspaces are typically used as integration areas, because 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 Configuring transfer transactions are performed from the perspective of the child workspace; the Putback transaction "puts back" files to the parent from the child workspace. See also *Bringover Create, Bringover Update,* and *Workspace*.

Putback Validation A Configuring feature that allows you to control which putbacks are allowed to a specific workspace.

Reparent To change the parent of a child workspace.

Resolve To produce a new delta of a file from two conflicting deltas. See *Merge* and *Conflict*.

Root directory The top-level directory of a Configuring workspace. This directory's path name is the name by which the workspace is referred.

RTI Request To Integrate, the heading used to record an Integration Request Identifier in a workspace history 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 increased automatically.

SCCS file properties Properties that you can assign to individual files.

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*.

SCCS Mergeable ID (SMID) A SMID is a number generated using the Xerox Secure Hash Function that ensures that every delta is uniquely identifiable, even if its SID is changed.

SID See SCCS delta ID.

SMID See SCCS Mergeable ID.

Uncheckout To return a file to the state it was in before the most recent check out.

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.

Version When you check in a file, SCCS records only the line-by-line differences between the text you check in and the previous version of the file. This set of differences is known as a delta. The file version that you initially checked out was constructed from a set of accumulated deltas. The terms delta and version are often used synonymously; however, their meanings are not the same. It is possible to retrieve a version that omits selected deltas. See *Delta*.

Versioning The Sun WorkShop TeamWare tool you use to manage Sun WorkShop TeamWare files. See Chapter 5.

Workspace A workspace is a specially designated directory, its subdirectories, and the files contained in those directories. Usually each developer on a project works in their own isolated workspace concurrently with other developers programming in other workspaces. Configuring provides utilities to manage workspaces. See Chapter 2.

Workspace hierarchy A hierarchy of parent and child workspaces in which programmers and release engineers can develop, test, share, and release software products.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z