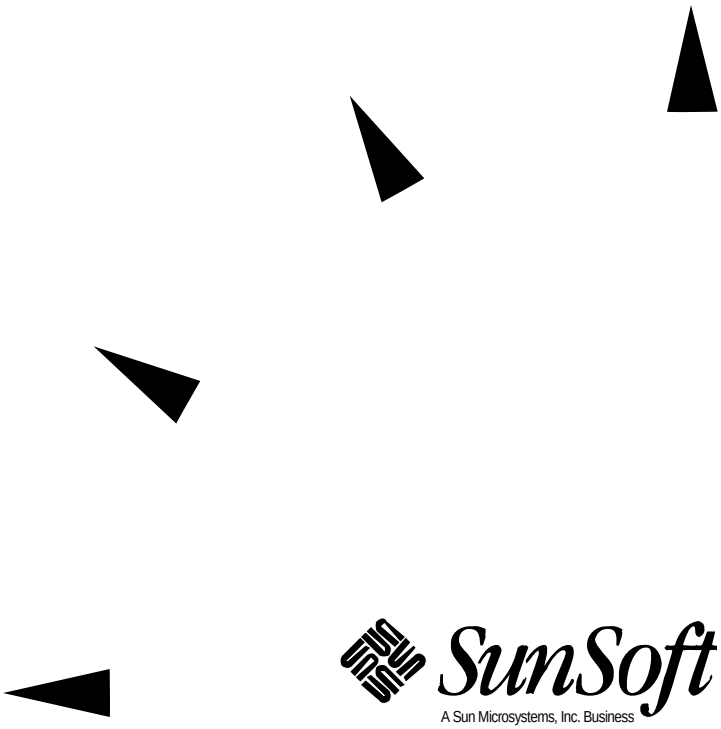
Solstice DiskSuite 4.0
Administration Guide



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

Part No: 802-2422-10
Revision A, March 1995

SunSoft
A Sun Microsystems, Inc. Business

Please
Recycle

Adobe PostScript

Contents

Figures

Preface

Solstice DiskSuite™ is an unbundled software package that offers a pseudo device driver (called a *metadisk driver*) providing better performance, greater capacity, and improved availability of data.

Solstice DiskSuite allows for up to three-way mirroring of any file system including `/usr`, `root`, and `swap`. Other features of Solstice DiskSuite include online concatenation of physical drives, online expansion of file systems, disk striping, hot spares, UFS logging, and RAID Level 5 support. Also included is a diskset feature which provides facilities for hosts to share disks in a high availabilty environment.

Solstice DiskSuite 4.0 runs on all SPARC® systems running Solaris™ 2.3 or a later Solaris 2.*x* release, and on all x86 systems running Solaris 2.4 or a later Solaris 2.*x* release. To use the UFS logging facilility, you must be running Solaris 2.4 or a later Solaris 2.*x* release.

The diskset feature is supported on SPARC Solaris 2.4 or a later SPARC Solaris 2.*x* release. This feature is not supported on x86 systems.

Who Should Use This Book

System administrators and others with the task of administering disk configurations and maintenance will find this manual to be a valuable resource. Much of the information in this book is targeted towards administrators with experience performing disk maintenance.

How This Book Is Organized

This document has an introduction, twelve chapters, and six appendixes. Examples are included at the end of chapters, where appropriate.

Chapter 1, "Introduction," introduces you to the basic concepts of Solstice DiskSuite, lists the peripherals supported, and defines some of the general terminology used in this document.

Chapter 2, "Installation and Setup," provides information on what you need to do before using Solstice DiskSuite, including installation and initial setup instructions for the software. This chapter also describes how to create the initial state database.

Chapter 3, "Overview of Solstice DiskSuite," offers a high-level overview of the functionality included with Solstice DiskSuite. The interaction between the various parts of Solstice DiskSuite is also discussed.

Chapter 4, "Concatenating and Striping," provides conceptual details and procedures for using the Solstice DiskSuite facilities that enable you to create metadevices consisting of either concatenated or striped disk partitions.

Chapter 5, "Mirroring," includes information on the operation of mirrors, using mirrors to recover from a single-component failure, the reconfiguration of mirrors, and the use of mirroring for online backups of file systems.

Chapter 6, "UFS Logging," provides information on how to set up and use the UNIX file system logging facility included with Solstice DiskSuite.

Chapter 7, "Hot Spares," describes how to define and use hot spares and includes information on conditions to avoid when using hot spares.

Chapter 8, "Disksets," provides information on how to set up and use the diskset utility included with Solstice DiskSuite. This feature provides facilities for hosts to share disks in a high availabilty environment.

Chapter 9, "RAID Devices," provides information on RAID (Redundant Arrays of Inexpensive Disks) Level 5, including hardware and software considerations and examples for the RAID 5 support in software.

Chapter 10, "State Database Replicas," provides an overview of the state database and discusses proper use.

Chapter 11, "Expanding a File System," provides instructions for expanding mounted and unmounted UFS file systems.

Chapter 12, "Configuration Guidelines," provides performance, capacity, and availability configuration tips.

Appendix A, "Solstice DiskSuite Files," discusses the format of the three system files associated with Solstice DiskSuite.

Appendix B, "Solstice DiskSuite Messages," explains the most common Solstice DiskSuite error messages.

Appendix C, "Recovery From Failed Boots," provides detailed information on how to recover from failed boots and other common errors.

Appendix D, "Upgrading to Other Solaris Versions," outlines a procedure for upgrading Solstice DiskSuite configurations to later versions of Solaris. This procedure is provided as an alternative to completely reinstalling the Solaris and Solstice DiskSuite packages.

Appendix E, "Using Solstice DiskSuite 4.0 with the SPARCstorage Array 100," describes the characteristics of the SPARCstorage Array (SSA) with which DiskSuite users should be familiar and explains how DiskSuite should be configured and used to take advantage of the SSA.

Appendix F, "Man Pages," is a printed copy of the associated manual pages, which are also included on the installation CD.

Related Books

Sun documentation related to the Solstice DiskSuite product and disk maintenance and configuration includes the following:

- *Solstice DiskSuite Tool 4.0 User's Guide*
- *Solaris 2.4 File System Administration*

What Typographic Changes and Symbols Mean

The following table describes the type changes and symbols used in this book.

Table P-1 Typographic Conventions

Typeface or Symbol	**Meaning**	**Example**
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`% su` `password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename*`.`
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.
Code samples are included in boxes and may display the following:		
`%`	UNIX C shell prompt	`%`
`$`	UNIX Bourne shell prompt	`$`
`#`	Superuser prompt, either shell	`#`

Introduction 1

The Solstice DiskSuite software product offers better performance, greater capacity, easier administration, and improved availability of disk storage on SPARC and x86 systems.

With Solstice DiskSuite, data availability and reliability are improved with three-way mirroring. You can mirror `swap` or any file system, including `root` and `/usr`. You can mirror existing file systems and automatically replace failed components within a mirror using hot spare facilities. Resynchronization time for mirrors is greatly reduced by optimized copying.

The UNIX file system (UFS) logging facility provides faster local directory operations, speeds up reboots, and decreases synchronous disk writes by recording file system updates in a log before they are applied to the UFS file system.

In addition, DiskSuite's disk striping and concatenation can increase performance. Striping spreads data requests over multiple components. Concatenation and stripes increase capacity by grouping several components into a single large logical device.

The information about the configuration and state of all metadevices is preserved in a replicated state database. This further ensures the integrity of data.

Administration is simplified by the hot spare facility and the dynamic growth of metadevices and file systems. New functionality included in the Solstice DiskSuite 4.0 release includes the creation of RAID (level 5) configurations and

disksets. A diskset is a grouping of two hosts and disk drives in which all the drives are accessible by both hosts. RAID provides recovery from disk failure in a more cost effective way than disk mirroring.

This introductory chapter provides the following:

- Information on how to get help
- A list of the peripherals supported by Solstice DiskSuite
- A list of DiskSuite terms and their definitions

Getting Help

If you have problems installing or using Solstice DiskSuite, call the distributor from which you purchased the software and provide the following information:

- Your name and electronic mail address (if available)
- Your company name, address, and phone number
- The model and serial number of your system
- The release number of the operating system (for example, Solaris 2.3 or Solaris 2.4)
- Any additional information that will help diagnose the problem

Supported Peripherals

Solstice DiskSuite 4.0 runs on all SPARC systems running Solaris 2.3 or a later Solaris 2.*x* release and on all x86 systems running Solaris 2.4 or a later Solaris 2.*x* release. (You must run Solaris 2.4 or a later Solaris 2.*x* release to use the UFS logging facility.)

The peripherals supported by DiskSuite include:

- IPI disk drives
- SCSI disk drives
- IDE disk drives (x86 only)

Note – The root file system cannot be mirrored on an IDE drive.

Disk Naming

Backup Copilot uses the normal disk naming convention for all drives. This naming is, `c`*n*`t`*n*`d`*n*`s`*n*, where `c` is the controller number, `t` is the target, `d` is the disk, and `s` is the slice. However it is important to be aware that on x86 systems, IDE drives do not have a target number. All the examples and instructions in this manual are from a SPARC system and show the target number.

Definition of Terms

The following are general definitions for Solstice DiskSuite:

Attach submirror - to add a submirror to an existing metamirror by using the `metattach(1M)` command. The attached submirror is then resynced with other submirrors.

Attach logging device - to add a logging device to an existing metatrans device by using the `metattach(1M)` command.

Component - the physical partition that is part of a metadevice.

Concatenated Stripe - a metadevice that is made up of both concatenated and striped components.

Concatenation - the act of combining two or more physical components into a single metadevice. The partitions are accessed sequentially (treated as a single device) rather than interlaced (as with *stripes*).

Detach submirror - to remove a submirror from an existing metamirror by using the `metadetach(1M)` command.

Detach logging device - to remove a logging device from a metatrans device by using the `metadetach(1M)` command.

Diskset - a grouping of two hosts and disk drives in which all the drives are accessible by both hosts. This feature provides facilities for hosts to share disks in a high availabilty environment.

Hot spare - a component set up to be automatically substituted for a failed component of a mirrored or RAID metadevice.

Hot spare pool - a group of spare components which automatically replace failed components.

Interlace - the number of blocks on a component of a striped or RAID metadevice that can be accessed simultaneously with the same number of blocks from other components. The interlace value tells DiskSuite how much data to place on a component of a striped or RAID metadevice before moving on to the next component.

Local diskset - a diskset for a host consisting of all drives which are not part of a shared diskset.

Logging - recording UNIX file system (UFS) updates in a log (the *logging device*) before they are applied to the UNIX file system (the *master device*).

Logging device - the component of a metatrans device that contains the log.

Master device - the component of a metatrans device that contains the file system.

Metadevice - a group of components accessed as a single logical device through concatenating, striping, mirroring, logging the physical devices, or setting up RAID devices.

Metadevice state database - information kept in nonvolatile storage (on disk) for preserving the state and configuration of metadevices.

Metadriver - a pseudo device driver that maps operations on a metadevice into operations on its components.

Metamirror - a special type of metadevice (also referred to as a “Mirror”), composed of one or more other metadevices called submirrors.

Metatrans device - a special type of metadevice which implements UFS logging (also referred to as a “Trans device”). A metatrans device is composed of one or more other metadevices or components. This pseudo device is made up of a master device and, optionally, a logging device.

md.tab - the metadevice workspace file for `metainit`. Note that this file is maintained by the user.

Mirroring - replicating all writes to a single logical device (the metamirror) to multiple devices (the submirrors) while distributing read operations. This provides redundancy of data in the event of a failure.

Optimized resync - an update of only the submirror regions that are not in sync when the system reboots. Because DiskSuite keeps track of regions of all metamirrors, the metadisk driver knows which regions of the submirrors are not identical (in sync) on all submirrors after a crash.

Partial resync - resyncing only a replaced component of a submirror or RAID device, rather than the entire submirror or RAID device.

RAID - an acronym for Redundant Arrays of Inexpensive Disks.

Replica - a copy of the state database. Keeping copies of the state database protects against the loss of state and configuration information. This information is critical to the operation of all metadevices.

Resync region - a division of a metamirror that enables DiskSuite to track changes by region rather than over the entire metamirror. By dividing the metamirror into regions, resync time may be reduced.

Resyncing - copying data from one submirror to another after system crashes, submirror failures, or after using the `metattach` command to add a submirror. Resyncing ensures the data on all submirrors is identical.

State database - a dedicated portion of a disk reserved exclusively for the metadisk driver. It cannot be used for any other purpose.

Stripe - similar to concatenation, except the addressing of the component blocks is interlaced on the partitions, rather than addressed sequentially. Striping is used to gain performance. By striping data across disks, multiple controllers can access data simultaneously.

Submirror - a metadevice that is part of a metamirror.

UFS - an acronym for the UNIX file system.

UFS logging - recording UNIX file system updates to a log (the logging device) before the updates are applied to the UFS (the master device).

1

Installation and Setup 2

This chapter provides the information necessary to perform the basic Solstice DiskSuite 4.0 installation.

It is assumed that you have a CD-ROM drive already installed on a machine. For a full description of the Solaris 2.4 system software installation, refer to *Installing Solaris Software.*

The basic steps to install software from CD-ROM include:

- Making sure you have enough space for the packages
- Adding the software packages
- Setting the system path variables

Use the following table to locate specific information in this chapter.

2

Converting to Solstice DiskSuite 4.0

The format of the metadevice state database replicas in Solstice DiskSuite 4.0 is incompatible with that of Online: DiskSuite 3.0 (or earlier DiskSuite versions). Because of this, Solstice DiskSuite 4.0 will not recognize or read database replicas created by Online: DiskSuite 3.0. This means that you must completely de-install any previous version of Online: DiskSuite before installing Solstice DiskSuite 4.0. The procedures provided in the following sections will guide you in this process.

If no version of DiskSuite has not been previously installed on this system, skip ahead to “Preparing for Installation” on page 11.

Before You Begin the Installation

Complete the following procedure *before* you start the installation procedures for Solstice DiskSuite 4.0.

1. **Back up all filesystems that currently reside on metadevices.**
   Note that this is only to ensure that no data is lost during the installation process. You will most likely not have to restore your filesystems from these backups.

2. **Save the existing state information.**
   You’ll need to create a new `/etc/opt/SUNWmd/md.tab` file from the state information displayed by the `metastat` command. Use the `metastat` command with the `-p` option to create an `md.tab` file. You must examine the `metastat` output to discover which, if any, devices are currently in “`Needs Maintenance`” state. This initial `md.tab` file will need to be edited, depending on the state of your metadevices.

3. **Edit the `md.tab` file.**
   You’ll need to make the following changes to the initial `md.tab` file:

- Use the `metastat` command with no arguments to determine which (if any) metadevices are in the “`Needs Maintenance`” state.

- Remove any submirrors that are in the "Needs Maintenance" state. These should be fixed and then attached (using metattach) after Solstice DiskSuite 4.0 is installed, and after the initial mirror is created.

Note – If both submirrors in a mirror are in the "Needs Maintenance" state, you will not be able to use this mirror with Solstice DiskSuite 4.0 until at least one of the submirrors is repaired.

- Delete all but the first submirror in the "OK" state from any remaining mirror lines. All mirror lines in the md.tab file should be in the form:

```
name -m submirror [ pass_number ]
```

Note that all mirrors are one-way mirrors. These will be created, and other submirrors attached, after the installation is complete.

4. **Record the location of all existing metadevice state database replicas in the** md.tab **file**
   The individual lines should be of the form:

```
mddb<nnn> slice1 [ slice2 slice3 ... ]
```

You can determine the location of all existing state database replicas by using the command metadb -i. Be careful to note the possible existence of multiple database replicas per device. Please refer to the md.tab(4) man page for the correct syntax of the mddb lines.

5. **Save the resulting** md.tab **file under a different name so it is not overwritten when you install DiskSuite 4.0.**
   You may want to copy the md.tab file to a floppy or another machine to be absolutely certain that no data is lost.

6. **Detach the log from all metatrans devices**
   You can use the metadetach command to detach the log. This flushes all data from the log onto the master device. Clearing the metatrans device with metaclear will accomplish this also, but the metatrans device will have to be unmounted before it can be cleared.

7. **Unmount all of the file systems that are currently mounted on metadevices, and comment entries for metadevices out of** `/etc/vfstab`.
   If `/`, `swap`, or `/usr` are mirrored, you will need to un-mirror those devices and reboot onto physical devices before proceeding. For more information on performing these procedures, see Chapter 5, "Mirroring." Similarly, if `/usr` is logged (is a metatrans device), you will need to "unlog" it by detaching the log and rebooting onto a physical device before proceeding.

8. **Delete all metadevice state database replicas using the** `metadb` **command with the** `-df` **options.**

9. **Remove the product by typing**:

```
# pkgrm SUNWmd
```

   Note that you must be superuser to run this command.

10. **Reboot the machine.**
    You have now completed the process for removing the previous version of Online: DiskSuite.

11. **Install Solstice DiskSuite 4.0 according to the instructions under the sections for local or remote installation.**
    After the installation is complete, reboot the machine and proceed with the post-installation instructions given below.

After the Installation

When you are finished installing Solstice DiskSuite 4.0, complete the following procedure to reinstate your former DiskSuite configuration.

1. **Copy the** `md.tab` **file you saved in the previous section to** `/etc/opt/SUNWmd/md.tab`**.**
   You may wish to run `metainit -n` to check the syntax of your `md.tab` file.

2. **Re-create all of your state database replicas using the** `metadb` **command with the** `-a` **option for each of the** `mddb<nnn>` **entries in your** `md.tab` **file as follows.**
   Note that, for the first replica, you will also need to add the `-f` option:

```
# metadb -a -f mddb<nnn>
```

   Note that if you want multiple database replicas per device, you must use the `-c` option to `metadb`.

3. **Re-create the metadevices using** `metainit -a`**.**

4. **Un-comment the entries in** `/etc/vfstab` **for metadevices that you commented out in step 6 above. Use the** `mount -a` **command to remount these file systems.**

5. **Reboot.**
   You should now be running the same metadevice configuration as you were before beginning the installation, except that you will have only one-way mirrors.

6. **Use** `metattach` **to attach the submirrors to all mirrors that previously had two (or more) submirrors.**
   If you have lots of these, it may be more convenient to prepare a shell script to perform the `metattach` operations.

Preparing for Installation

The following sections describe how to get ready for a local or remote DiskSuite installation.

Local Installation

To prepare for a local installation:

1. **Insert the CD containing the software into a caddy. Then put the caddy into the CD-ROM drive.**

2. **Change directories to** `/cdrom/cdrom0` **as follows:**

```
local% cd /cdrom/cdrom0
```

You are now ready to install the software on your local machine. Skip to the section "Adding Packages" on page 13 and follow the instructions provided.

Remote Installation

To prepare for a remote installation:

1. **On the remote machine, insert the CD containing the software into a caddy. Then put the caddy into the CD-ROM drive.**
2. **Put a line similar to the following into the file** `/etc/dfs/dfstab`**:**

```
share -F nfs -o ro -d "CD-ROM Directory" /cdrom/cdrom0
```

This line may be different, depending on how your system is networked.

3. **Export the** `/cdrom/cdrom0` **directory with the** `shareall` **command:**

```
remote# shareall
```

4. **On the local machine, log in as root and create the directory** `/cdrom/cdrom0` **(if it doesn't already exist):**

```
local% su
Password: root-password
local# mkdir -p /cdrom/cdrom0
```

You may choose another directory besides `/cdrom/cdrom0`.

5. **Mount the CD-ROM as follows:**

```
local# mount remote_machinename:/cdrom/cdrom0 /cdrom/cdrom0
```

You're now ready to install the software onto your local machine. Skip to the section "Adding Packages" on page 13 and follow the instructions provided.

Adding Packages

1. **Become root (if you haven't already).**

```
local% su
Password: root-password
```

2. **Change to the directory on which the CD-ROM is mounted.**
   (The section, "Mounting the CD-ROM," called this directory `/cdrom/cdrom0`. You may have chosen a different name.)

```
local# cd /cdrom/cdrom0
```

3. **Run `pkgadd` to install packages.**

```
local# pkgadd -d .
```

Note – If the `pkgadd` command is not in your current path, you must specify the full path to the command (`/usr/sbin/pkgadd`).

4. **Choose the package you want to install.**
   `pkgadd` displays the available packages and prompts you to enter the number associated with a package. Select `1` to install the AnswerBook® online documentation only (see "Viewing AnswerBook Online Documentation" on page 22 for details on how to proceed), select `2` to install the DiskSuite package only, or select '`all`' (as illustrated in the following example) to install both packages. The program loops until you press `q` to quit.

Note – Do not be concerned if the screens displayed when you install this product do not appear exactly as shown in the following example.

```
# pkgadd -d .

The following packages are available:
  1  SUNWabmd  Solstice DiskSuite 4.0 Administration Guide AnswerBook
                (all) 24.2.7
  2  SUNWmd    Solstice DiskSuite
                (all) 4.0,REV=1.0
  3  SUNWmdg   Solstice DiskSuite Tool
                (all) 4.0,REV=1.0

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]: all

Processing package instance <SUNWabmd> from </cdrom>

Solstice DiskSuite 4.0 Administration Guide AnswerBook
(all) 24.2.7
     Copyright 1994 Sun Microsystems, Inc. All Rights Reserved
          Printed in the United States of America
2550 Garcia Avenue, Mountain View, California, 94043-1100 U.S.A.

... (miscellaneous copyright information)...

Using </opt> as the package base directory.

The installation options are as follows:
Option: Description:
-------------------------------------------
1. nil:    less than 1 Megabyte disk space required [slowest
performance].
2. heavy:  1.88 Megabytes disk space required [best performance].

Note: If the install option which you choose fails
      due to lack of space, try another location, or
      choose a lower install option number.
Enter the number of an installation option from the list above (1 or
2):

Select an installation option: 2
Installation option: heavy selected.

       (continued on following page)
```

```
Specify the parent of the AnswerBook home directory:
 For the heavy option all files will be placed under /opt/SUNWabmd.
## Processing package information.
## Processing system information.
## Verifying package dependencies.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.
## Checking for setuid/setgid programs.

This package contains scripts which will be executed with super-user
permission during the process of installing this package.

Do you want to continue with the installation of this package [y,n,?] y

Installing Solstice DiskSuite 4.0 Administration Guide AnswerBook as
<SUNWabmd>

## Installing part 1 of 1.
/opt/SUNWabmd/spot-help/README
[ verifying class <HelpFiles> ]
/opt/SUNWabmd/_data/DISKSUITEADMIN/01.Introduction
/opt/SUNWabmd/_data/DISKSUITEADMIN/02.Getting_Started
/opt/SUNWabmd/_data/DISKSUITEADMIN/03.Overview_of_Sun_OnlineDiskSuite
/opt/SUNWabmd/_data/DISKSUITEADMIN/04.Concatenating_and_Striping
/opt/SUNWabmd/_data/DISKSUITEADMIN/05.Mirroring
/opt/SUNWabmd/_data/DISKSUITEADMIN/06.UFS_Logging
/opt/SUNWabmd/_data/DISKSUITEADMIN/07.Hot_Spares
/opt/SUNWabmd/_data/DISKSUITEADMIN/08.Disksets
/opt/SUNWabmd/_data/DISKSUITEADMIN/09.RAID_Devices
/opt/SUNWabmd/_data/DISKSUITEADMIN/10.State_Database_Replicas
/opt/SUNWabmd/_data/DISKSUITEADMIN/11.Expanding_a_File_System
/opt/SUNWabmd/_data/DISKSUITEADMIN/12.Configuration_Guidelines
/opt/SUNWabmd/_data/DISKSUITEADMIN/A.Sun_Online_DiskSuite_Files
/opt/SUNWabmd/_data/DISKSUITEADMIN/B.Sun_Online_DiskSuite_...
/opt/SUNWabmd/_data/DISKSUITEADMIN/C.Recovery_From_Failed_Boots
/opt/SUNWabmd/_data/DISKSUITEADMIN/Contents
/opt/SUNWabmd/_data/DISKSUITEADMIN/Cover
/opt/SUNWabmd/_data/DISKSUITEADMIN/Credits
/opt/SUNWabmd/_data/DISKSUITEADMIN/D.Upgrading_to_Other_...
/opt/SUNWabmd/_data/DISKSUITEADMIN/E.Using_Solstice_DiskSuite...
/opt/SUNWabmd/_data/DISKSUITEADMIN/F.Man_Pages

       (continued on following page)
```

```
/opt/SUNWabmd/_data/DISKSUITEADMIN/Preface
/opt/SUNWabmd/_data/DISKSUITEADMIN/files_to_print
/opt/SUNWabmd/_data/SUNWab_24_1.xtoc
/opt/SUNWabmd/_data/SUNWab_24_1/ManPagesEntryPoint
/opt/SUNWabmd/_data/SUNWab_24_1/SUNWab_24_1.titlepage
/opt/SUNWabmd/_data/SUNWab_24_1/SystemLink.SUNWab_24_1
/opt/SUNWabmd/_data/SUNWab_24_1/abmerge
/opt/SUNWabmd/_data/SUNWab_24_1/abunmerge
/opt/SUNWabmd/_data/SUNWab_24_1/dbgen
/opt/SUNWabmd/_data/SUNWab_24_1/domerge
/opt/SUNWabmd/_data/SUNWab_24_1/initial.page
/opt/SUNWabmd/_data/SUNWab_24_1/make_ab_script
/opt/SUNWabmd/_data/SUNWab_24_1/xtocmerge
/opt/SUNWabmd/_data/list_of_books
/opt/SUNWabmd/_data/sr_index/Keys
/opt/SUNWabmd/_data/sr_index/SUNW24_1.cat
/opt/SUNWabmd/_data/sr_index/SUNW24_1.cfg
/opt/SUNWabmd/_data/sr_index/SUNW24_1.cix
/opt/SUNWabmd/_data/sr_index/SUNW24_1.dct
/opt/SUNWabmd/_data/sr_index/SUNW24_1.log
/opt/SUNWabmd/_data/sr_index/SUNW24_1.ref
/opt/SUNWabmd/_data/sr_index/ascii.xtoc
[ verifying class <PostScript> ]
## Executing postinstall script.
----- Installing  2  Book Contents databases -----

The destination for the Book Contents databases is
/opt/SUNWabmd/_data

After installation, use /opt/SUNWabmd/answerbook to start the AB.

Installation of <SUNWabmd> was successful.
Processing package instance <SUNWmd> from </cdrom>

Solstice Disksuite
(sparc i386) OLDS_4.0
Solstice DiskSuite 4.0 SUNBIN
CD-ROM (HSFS Format)
Part Number 258-3523-10

Copyright 1994 Sun Microsystems, Inc.
2550 Garcia Avenue, Mountain View, California, 94043-1100 U.S.A.
```

(*continued on following page*)

```
All rights reserved... (miscellaneous copyright information)...

Using </> as the package base directory.
## Processing package information.
## Processing system information.
   4 package pathnames are already properly installed.
## Verifying package dependencies.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.
## Checking for setuid/setgid programs.

This package contains scripts which will be executed with super-
user permission during the process of installing this package.

Do you want to continue with the installation of this package
[y,n,?] y

Installing Solstice DiskSuite as <SUNWmd>

##Executing preinstall script.
## Installing part 1 of 1.
/etc/init.d/SUNWmd.init
/etc/init.d/SUNWmd.sync
/etc/rc2.d/S95SUNWmd.sync <symbolic link>
/etc/rcS.d/S35SUNWmd.init <symbolic link>
/kernel/drv/md
/kernel/drv/md.conf
/kernel/misc/md_hotspares
/kernel/misc/md_mirror
/kernel/misc/md_stripe
/usr/lib/drv/preen_md.so.1
/usr/opt/SUNWmd/etc <symbolic link>
/usr/opt/SUNWmd/locale/C/LC_MESSAGES/SUNWmd.po
/usr/opt/SUNWmd/man/man1m/growfs.1m
/usr/opt/SUNWmd/man/man1m/metaclear.1m
/usr/opt/SUNWmd/man/man1m/metadb.1m
/usr/opt/SUNWmd/man/man1m/metadetach.1m
/usr/opt/SUNWmd/man/man1m/metahs.1m
/usr/opt/SUNWmd/man/man1m/metainit.1m
/usr/opt/SUNWmd/man/man1m/metaoffline.1m
/usr/opt/SUNWmd/man/man1m/metaonline.1m
/usr/opt/SUNWmd/man/man1m/metaparam.1m

(continued on following page)
```

```
/usr/opt/SUNWmd/man/man1m/metareplace.1m
/usr/opt/SUNWmd/man/man1m/metaroot.1m
/usr/opt/SUNWmd/man/man1m/metastat.1m
/usr/opt/SUNWmd/man/man1m/metasync.1m
/usr/opt/SUNWmd/man/man1m/metattach.1m
/usr/opt/SUNWmd/man/man4/md.cf.4
/usr/opt/SUNWmd/man/man4/md.tab.4
/usr/opt/SUNWmd/man/man4/mddb.cf.4
/usr/opt/SUNWmd/man/man7/md.7
/usr/opt/SUNWmd/sbin/growfs
[verifying class <none>]
cp /cdrom/SUNWmd/reloc/etc/opt/SUNWmd/md.cf
/etc/opt/SUNWmd/md.cf
cp /cdrom/SUNWmd/reloc/etc/opt/SUNWmd/md.tab
/etc/opt/SUNWmd/md.tab
cp /cdrom/SUNWmd/reloc/etc/opt/SUNWmd/mddb.cf
/etc/opt/SUNWmd/mddb.cf
cp /cdrom/SUNWmd/reloc/kernel/drv/md.conf /kernel/drv/md.conf
[ verifying class <preserve> ]
/kernel/drv/md
/kernel/misc/md_hotspares
/kernel/misc/md_mirror
/kernel/misc/md_stripe
/kernel/misc/md_trans
/usr/lib/drv/preen_md.so.1
/usr/opt/SUNWmd/sbin/metaclear
/usr/opt/SUNWmd/sbin/metadb
/usr/opt/SUNWmd/sbin/metadetach
/usr/opt/SUNWmd/sbin/metahs
/usr/opt/SUNWmd/sbin/metainit
/usr/opt/SUNWmd/sbin/metaoffline
/usr/opt/SUNWmd/sbin/metaonline
/usr/opt/SUNWmd/sbin/metaparam
/usr/opt/SUNWmd/sbin/metareplace
/usr/opt/SUNWmd/sbin/metaroot
/usr/opt/SUNWmd/sbin/metastat
/usr/opt/SUNWmd/sbin/metasync
/usr/opt/SUNWmd/sbin/metattach
[verifying class <sparc>]
```

(*continued on following page*)

```
## Executing postinstall script.
(various other manpage, command, help, and gui files installed here)
.
.
.
Installation of <SUNWmd> was successful.
.
.
.
Installation of <SUNWmdg> was successful.
The following packages are available:
  1  SUNWabmd   Solstice DiskSuite 4.0 Administration Guide AnswerBook
                (all) 24.2.7
  2  SUNWmd     Solstice DiskSuite
                (all) 4.0,REV=1.0
  3  SUNWmdg    Solstice DiskSuite Tool
                (all) 4.0,REV=1.0

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]: q

*** IMPORTANT NOTICE ***
       This machine must now be rebooted in order to ensure
       sane operation.  Execute shutdown -y -i6 -g0
       and wait for the “Console Login:” prompt.
```

5. **Respond with a** `y` **to any prompts about changing modes on directories.**
   `pkgadd` installs the DiskSuite files in the `/usr/opt/SUNWmd` directory, as well as in other system directories. `pkgadd` does not overwrite any software included with the standard Solaris 2.4 release.

6. **Reboot the system as directed to build the metadevices.**

2

Viewing Installed Packages

You can confirm that the DiskSuite software and AnswerBook have been installed by using the `pkginfo` command:

```
demo$ pkginfo
application SUNWabmd       AnswerBook
SPARCompile SUNWC++        SPARCompilers 2.0  C++ 3.0
system      SUNWaccr       System Accounting, (Root)
system      SUNWaccu       System Accounting, (Usr)
SPARCompile SUNWacomp      SPARCompilers 2.0  Sun C 2.0
system      SUNWarc        Archive Libraries
system      SUNWast        Automated Security Enhancement Tools
system      SUNWaudio      Audio applications
system      SUNWbcp        Binary Compatibility
system      SUNWbnur       Networking UUCP Utilities, (Root)
system      SUNWbnuu       Networking UUCP Utilities, (Usr)
system      SUNWmd         Solstice DiskSuite
...
```

The `-l` option gives detailed information about packages:

```
demo$ pkginfo -l SUNWmd
   PKGINST:  SUNWmd
      NAME:  Solstice DiskSuite
  CATEGORY:  system
      ARCH:  sparc_and_i386
   VERSION:  SDS_4.0
   BASEDIR:  /
    VENDOR:  SunSoft, a Sun Microsystems, Inc. Business
      DESC:  SunSoft's Solstice DiskSuite
    PSTAMP:  ########
  INSTDATE:  Jan 10 1995 11:26
   HOTLINE:  Please contact your local service provider
    STATUS:  completely installed
     FILES:     70 installed pathnames
                10 shared pathnames
                22 directories
                21 executables
                11 package information files
              1929 blocks used (approx)
...
```

Removing Packages

If you decide to remove either the DiskSuite or AnswerBook packages, simply remove the installed files using the `pkgrm` command. For example, to remove the AnswerBook package (as superuser), type:

```
# pkgrm SUNWabmd
```

Note – Do not remove DiskSuite or AnswerBook files using the `rm` command. Using `pkgrm` is the only valid way to remove these files.

Path Requirements

After installing the software, you must set the environment variables `PATH` and `MANPATH`.

The general requirements are as follows:

- Insert `/usr/opt/SUNWmd/sbin` either before or after `PATH`.

  If you do not set this path, you may get other versions of the software.

- Insert `/usr/opt/SUNWmd/man` either before or after your `MANPATH`.

  If you do not set this path, you could get man pages for the wrong release, or no man pages at all.

You can set these paths by using an editor to change your `$HOME/.profile` or `~/.cshrc` file, as follows.

If you installed into the default directory, then:

- If you're using the Bourne shell, your `$HOME/.profile` file should have lines like this:

```
PATH=/usr/opt/SUNWmd/sbin:$PATH
MANPATH=/usr/opt/SUNWmd/man:$MANPATH
export PATH MANPATH
```

- If you're using the C shell (csh), the `~/.cshrc` file should have lines that look something like this:

```
set path = (/usr/opt/SUNWmd/sbin $path)
setenv MANPATH /usr/opt/SUNWmd/man:$MANPATH
```

Viewing AnswerBook Online Documentation

To view the AnswerBook online documentation:

1. **Type** `/usr/openwin/bin/answerbook` **and press Return.**
2. **The AnswerBook Navigator is displayed.**
   Follow the online instructions to use the AnswerBook online documentation.

Creating Replicas of the Metadevice State Database

Before you can use the DiskSuite software, you must create the metadevice state database. The metadevice state database replicas can exist on dedicated disk partitions or within a concatenated, striped, or logging metadevice.

It is strongly recommended you make a replica on each available unused partition. There must be at least three replicas or the DiskSuite software can not be used. Read the following section for instructions on creating space for metadevice state database replicas.

Creating Space for Metadevice State Database Replicas

Before using any DiskSuite functionality, you must create at least three replicas (copies) of the metadevice state database. The three replicas ensure the loss of any one replica will not cause a failure of DiskSuite.

A replica is a dedicated portion of a disk, similar to a disk label. The space occupied by the replica is reserved for the exclusive use of the metadevice state database; it should not be used for any other purpose.

State database replicas are critical to the operation of all metadevices because they provide nonvolatile storage for DiskSuite. The replicas keep track of all configuration and status information about mirrors, submirrors, concatenations, stripes, hot spares, logging devices, and master devices. The replicas also keep track of error conditions.

The replicated state database information keeps DiskSuite operating. Without replicas of the same information for comparison, DiskSuite does not know the current running state of metadevices. Chapter 8, “Disksets,” and the `metadb (1M)` manual page discuss how the replicas are used by the metadisk driver.

Each replica may exist on either a dedicated disk partition or within space specifically reserved for a replica within a striped or concatenated metadevice; or within a logging or a RAID device. Multiple replicas can be stored in a single disk partition. However, placing all replicas on a single disk reduces reliability. Each replica occupies 517 kilobytes (Kbytes) or 1034 disk blocks of the partition by default.

Note – Before you create replicas, make sure you have a current backup of all data.

There are three procedures you can use for creating replicas. These procedures are:

- Creating replicas on metadevice components. Use this procedure if you have at least three unused components on which you can create replicas.
- Creating a new disk partition. Use this procedure if you have no unused disk partitions and are forced to repartition the disk to create the space for replicas.
- Creating replicas on unused disk partitions. Use this procedure if you have only one unused disk partition on which you can create replicas.

The first and third procedures on this list are actually the preferred methods for creating replicas. The second procedure is described in the event neither of the other two can be used at your site. The following sections detail the three procedures.

Creating Replicas on Metadevice Components

This section describes how to create three replicas on unused components that will be used within a metadevice. Unused components are disk partitions that have no active data on them (for example, no file systems, `swap`, or databases reside on the partition).

For a detailed discussion of what metadevices are and how you would use them at your site, refer to Chapter 3, "Overview of Solstice DiskSuite.''

Note – If you plan to use the components within a new metadevice, consider placing replicas on at least three of the physical components of the new metadevice. This will prevent the failure of a single disk from causing DiskSuite to fail.

The following procedure describes how to create three replicas on three components that will be part of a metadevice.

- **Determine which components will be part of a metadevice and use the `metadb` command to create replicas on the components.**
  For instance, if you have three components, `/dev/dsk/c0t1d0s3`, `/dev/dsk/c1t1d0s3`, and `/dev/dsk/c2t1d0s3`, that you plan to concatenate or use as logging devices, you would enter the following as root:

```
# /usr/opt/SUNWmd/sbin/metadb -a -f /dev/dsk/c0t1d0s3    \
/dev/dsk/c1t1d0s3 /dev/dsk/c2t1d0s3
```

The `metadb` command creates one replica on each of the components, `/dev/dsk/c0t1d0s3`, `/dev/dsk/c1t1d0s3`, and `/dev/dsk/c2t1d0s3`.

When you run `metadb` for the first time, the following message is displayed:

```
metadb: There are no existing databases
```

You have just created the first state database replicas so you can ignore this message.

In this case, running `metadb` creates empty replicas. The replicas will not contain any configuration information until other DiskSuite utilities are used to manipulate metadevices.

Allocating Space From the `swap` *Partition for Replicas*

If there are no unused disk partitions, you can create a new partition by taking space from the end of the `swap` partition and assigning it to an unused partition name.

The procedure for allocating space from the `swap` partition is as follows:

1. As root, halt the system using the `halt` command.

```
# halt
Oct 29 14:23:56 demo halt: halted by root
Oct 29 14:23:58 1992
  demo syslogd: going down on signal 15
Halted

Program terminated
Type help for more information
```

2. Boot the system to single-user mode as follows:

```
<#3> ok boot -s
Resetting ...

SPARCsystem 600MP (4 X 390Z55), No Keyboard
ROM Rev. 2.8, 128 MB memory installed, Serial #4200929.
Ethernet address 8:0:20:10:cb:ee, Host ID: 714019e1.


Initializing Memory ...
Boot device: /iommu/sbus/dma@f,81000/esp@f,80000/sd@3,0   File
and args: -s
SunOS Release 5.1 Version beta_1.2 [UNIX(R) System V Release 4.0]
Copyright (c) 1983-1992, Sun Microsystems, Inc.
Hostname: demo

INIT: SINGLE USER MODE

Type Ctrl-d to proceed with normal startup,
(or give root password for system maintenance): <root-password>
Entering System Maintenance Mode

SunOS Release 5.1 Version beta_1.2 [UNIX(R) System V Release 4.0]
#
```

3. Turn off swapping to the partition from which you intend to use blocks.
For example:

```
# swap -l
swapfile             dev  swaplo blocks   free
swapfs                 -       0 751552 750136
/dev/dsk/c0t3d0s1   32,25      8 524952 524952
# swap -d /dev/dsk/c0t3d0s1
# swap -l
swapfile             dev  swaplo blocks   free
swapfs                 -       0 226600 225176
#
```

4. **Changing the disk partitions involves editing the disk label. Use the `format` command to repartition the disk.**
   For example:

```
# format
Searching for disks...done

AVAILABLE DISK SELECTIONS:
       0. c0t2d0 <SUN1.3G cyl 1965 alt 2 hd 17 sec 80>
          /iommu@f,e0000000/sbus@f,e0001000/esp@f,80000/sd@2,0
       1. c0t3d0 <SUN1.3G cyl 1965 alt 2 hd 17 sec 80>
          /iommu@f,e0000000/sbus@f,e0001000/esp@f,80000/sd@3,0
       2. c1t1d0 <SUN1.3G cyl 1965 alt 2 hd 17 sec 80>
          /iommu@f,e0000000/sbus@f,e0001000/esp@0,200000/sd@1,0
       3. c1t3d0 <SUN1.3G cyl 1965 alt 2 hd 17 sec 80>
          /iommu@f,e0000000/sbus@f,e0001000/esp@0,200000/sd@3,0
       4. c2t1d0 <SUN1.3G cyl 1965 alt 2 hd 17 sec 80>
          /iommu@f,e0000000/sbus@f,e0001000/esp@1,200000/sd@1,0
       5. c2t3d0 <SUN1.3G cyl 1965 alt 2 hd 17 sec 80>
          /iommu@f,e0000000/sbus@f,e0001000/esp@1,200000/sd@3,0
Specify disk (enter its number): 1
selecting c0t3d0
[disk formatted]
Warning: Current Disk has mounted partitions.

FORMAT MENU:
        disk       - select a disk
        type       - select (define) a disk type
        partition  - select (define) a partition table
        current    - describe the current disk
        format     - format and analyze the disk
        repair     - repair a defective sector
        label      - write label to the disk
        analyze    - surface analysis
        defect     - defect list management
        backup     - search for backup labels
        verify     - read and display labels
        save       - save new disk/partition definitions
        inquiry    - show vendor, product and revision
        volname    - set 8-character volume name
        quit
```

5. **Next, use the** `partition` **command to delete the cylinders from the** `swap` **partition that you intend to use for your database.**
   For example, to delete six cylinders from `swap` partition 1 to use for a database on slice 3, you would do the following:

```
format> partition

PARTITION MENU:
        0      - change ‘0' partition
        1      - change ‘1' partition
        2      - change ‘2' partition
        3      - change ‘3' partition
        4      - change ‘4' partition
        5      - change ‘5' partition
        6      - change ‘6' partition
        7      - change ‘7' partition
        select - select a predefined table
        modify - modify a predefined partition table
        name   - name the current table
        print  - display the current table
        label  - write partition map and label to the disk
        quit
partition> print
Current partition table (original sd3):
Part      Tag    Flag     Cylinders        Size       Blocks
  0       root    wm       0 -  150      100.27MB    (151/0/0)
  1       swap    wu     151 -  536      256.33MB    (386/0/0)
  2 unassigned    wm       0               0         (0/0/0)
  3 unassigned    wm       0               0         (0/0/0)
  4 unassigned    wm       0               0         (0/0/0)
  5          -    wm     537 -  838      200.55MB    (302/0/0)
  6        usr    wm     839 - 1290      300.16MB    (452/0/0)
  7          -    wm    1291 - 1964      447.58MB    (674/0/0)

partition> 1
Part      Tag    Flag     Cylinders        Size       Blocks
  1       swap    wu     151 -  536      256.33MB    (386/0/0)

Enter partition id tag[swap]: <RETURN>
Enter partition permission flags[wu]: <RETURN>
Enter new starting cyl[151]: <RETURN>
Enter partition size[524960b, 386c, 256.33mb]: 380c
```

6. Add the cylinders to the new partition.
Continuing the example from the previous step, you would add the six cylinders to the new partition 3 as follows:

```
partition> 3
Part      Tag    Flag     Cylinders        Size       Blocks
  3 unassigned    wm       0               0         (0/0/0)

Enter partition id tag[unassigned]: <RETURN>
Enter partition permission flags[wm]: <RETURN>
Enter new starting cyl[0]: 531
Enter partition size[0b, 0c, 0.00mb]: 6c
```

7. Verify the addition of the cylinders using the `print` command as follows:

```
partition> print
Current partition table (unnamed):
Part      Tag    Flag     Cylinders        Size       Blocks
  0       root    wm       0 -  150      100.27MB    (151/0/0)
  1       swap    wu     151 -  530      252.34MB    (380/0/0)
  2 unassigned    wm       0               0         (0/0/0)
  3 unassigned    wm     531 -  536        3.98MB    (6/0/0)
  4 unassigned    wm       0               0         (0/0/0)
  5          -    wm     537 -  838      200.55MB    (302/0/0)
  6        usr    wm     839 - 1290      300.16MB    (452/0/0)
  7          -    wm    1291 - 1964      447.58MB    (674/0/0)
```

8. Write the new label to the disk.

```
partition> label
Ready to label disk, continue? y
```

9. **Exit from the** `partition` **menu and then the** `format` **program by typing the** `quit` **command twice.**
   For example:

```
partition> quit

FORMAT MENU:
        disk       - select a disk
        type       - select (define) a disk type
        partition  - select (define) a partition table
        current    - describe the current disk
        format     - format and analyze the disk
        repair     - repair a defective sector
        label      - write label to the disk
        analyze    - surface analysis
        defect     - defect list management
        backup     - search for backup labels
        verify     - read and display labels
        save       - save new disk/partition definitions
        inquiry    - show vendor, product and revision
        volname    - set 8-character volume name
        quit
format> quit
```

10. **Reboot the machine.**

```
# reboot
rebooting...
Resetting...
```

Warning – Re-allocating space for a new partition can be done only on the `swap` partition. Attempting this process on a file system will cause the loss of data.

11. Log back in to the system and check that the new partition exists.

```
demo console login: root
Password: <root-password>
Last login: Fri Oct 23 14:23:50 on console
SunOS Release 5.1 Version beta_1.2 [UNIX(R) System V Release 4.0]
you have mail

demo# prtvtoc /dev/rdsk/c0t3d0s0
* /dev/rdsk/c0t3d0s0 partition map
*
* Dimensions:
*     512 bytes/sector
*      80 sectors/track
*      17 tracks/cylinder
*    1360 sectors/cylinder
*    3500 cylinders
*    1965 accessible cylinders
*
* Flags:
*   1: unmountable
*  10: read-only
*
*                          First     Sector    Last
* Partition  Tag  Flags    Sector     Count    Sector  Mount
Directory
       0      2    00          0    205360    205359   /
       1      3    01     205360    516800    722159
       3      0    00     722160      8160    730319
       5      6    00     730320    410720   1141039   /opt
       6      4    00    1141040    614720   1755759   /usr
       7      6    00    1755760    916640   2672399   /export
```

12. Create the desired number of replicas on the new partition.
For example, to create three replicas on the new slice 3:

```
demo# /usr/opt/SUNWmd/sbin/metadb -c 3 -a -f /dev/dsk/c0t3d0s3
```

In this example, the `-c 3` is used to specify that three replicas of the database are being put on the same partition.

When you use `metadb` for the first time, the following message is displayed:

```
metadb: There are no existing databases
```

You have just created the first state database replicas so you can ignore this message.

Running `metadb` creates empty replicas. The replicas will not contain any configuration information until other DiskSuite utilities are used to manipulate metadevices.

Warning – It is strongly recommended that you do *not* place all replicas of the state database on the *same* partition. If the partition is corrupted or the device fails, all replicas will be lost and DiskSuite will lose access to all metadevices.

Creating Replicas on Unused Disk Partitions

Most disk partitions are likely to be much larger than what is required for a replica. To prevent the waste of large amounts of disk space, you could take space from one partition and allocate it to an unused disk partition name or a new disk partition.

However, if you plan to use this disk partition in a metadevice, you don't need to reallocate space to a new disk partition. You need only create replicas on the disk partitions, as shown in Step 4 in the following procedure.

For a detailed discussion of what metadevices are and how you would use them at your site, refer to the "Metadevices" section in Chapter 3.

An example of the procedure you would use for allocating disk space and creating a small disk partition (2 Mbytes) follows:

1. **Run `cat` on `/etc/vfstab` to view the currently mounted file systems.**
   For example:

```
# cat /etc/vfstab
#device          device          mount   FS   fsck  mount   mount
#to mount        to fsck         point  type pass  at boot options
/dev/dsk/c0t3d0s0 /dev/rdsk/c0t3d0s0 /     ufs   1     no      -
/dev/dsk/c0t3d0s6 /dev/rdsk/c0t3d0s6 /usr  ufs   2     no      -
/dev/dsk/c0t3d0s5 /dev/rdsk/c0t3d0s5 /opt  ufs   4     yes     -
/dev/dsk/c0t3d0s1 -                 -      swap  -     no      -
#
/proc             -                 /proc   proc  -     no      -
fd                -                 /dev/fd fd    -     no      -
swap              -                 /tmp          -     yes     -
```

2. **Use the `prtvtoc` command to find the names of available partitions on the device.** For example:

```
# prtvtoc /dev/rdsk/c0t3d0s0
/dev/rdsk/c0t3d0s0 partition map

Dimensions:
      512 bytes/sector
       80 sectors/track
       17 tracks/cylinder
     1360 sectors/cylinder
     3500 cylinders
     1965 accessible cylinders

Flags:
    1: unmountable
   10: read-only

                         First    Sector     Last
Partition  Tag  Flags    Sector    Count     Sector    Mount Directory
     0      2    00         0    205360    205359      /
     1      3    01    205360    524960    730319
     5      6    00    730320    410720   1141039      /opt
     6      4    00   1141040    614720   1755759      /usr
     7      6    00   1755760    916640   2672399
```

From the above output you can see that `/dev/dsk/c0t3d0s7` has space available, and you also know from the contents of `/etc/vfstab` (in Step 1) that `/dev/dsk/c0t3d0s7` is not in use as a file system or swap space. Partition `c0t3d0s7` could be used for replicas of the metadevice state database. However, partition `c0t3d0s7` is approximately 458 Mbytes (916,640 sectors at 512 bytes per sector). If you place three replicas on `c0t3d0s7`, you are wasting more than 450 Mbytes. This space could be reclaimed by placing a metadevice on this partition.

Because the partition names `c0t3d0s3` and `c0t3d0s4` are not being used (they do not correspond to any sector range on the disk) any one of these could be used for the database. If you take 2 Mbytes of space from `/dev/dsk/c0t3d0s7` and create a new partition, `c0t3d0s3`, containing this space, you can place the replicas there.

On this disk, each cylinder occupies 680 Kbytes, thus you can allocate one cylinder per replica.

3. **Next, use the `format` command to repartition the disk as shown in Step 4 of the previous section.**
   Make sure the disk space assigned to the partition you are creating for the metadevice state database is not shared with another partition. For example, a common partitioning scheme has the 3, 4, and 5 partitions overlapping the 6 partition. If you plan to put the state database on the 3 partition, make sure that the 6 partition is never used.

4. **Now that the space has been taken from `/dev/dsk/c0t3d0s7` and allocated to `/dev/dsk/c0t3d0s3`, create three replicas on the partition. To create the replicas, use the `metadb` command as follows:**

```
# /usr/opt/SUNWmd/sbin/metadb -c 3 -a -f /dev/dsk/c0t3d0s3
```

   In the above example, the `-c 3` specifies that three replicas of the database are being put on the same partition.

   When `metadb` is run for the first time, the following message is displayed.

```
metadb: There are no existing databases
```

   You have just created the first state database replicas so you can ignore this message.

   Running `metadb` creates empty replicas. The replicas will not contain any configuration information until other DiskSuite utilities are used to manipulate metadevices.

Warning – It is strongly recommended that you do *not* place all replicas of the state database on the *same* partition. If the partition is corrupted or the device fails, all replicas will be lost and DiskSuite will lose access to all metadevices.

What to Do Next

Now that the DiskSuite software is set up on your system, you can begin to use the new functionality provided. DiskSuite provides considerable new functionality. Read Chapter 3, “Overview of Solstice DiskSuite,” to familiarize yourself with the product.

2

Overview of Solstice DiskSuite 3

Solstice DiskSuite is an unbundled software package that offers enhanced availability, performance, capacity, reliability, and administration.

DiskSuite provides higher data availability and reliability by supporting the mirroring of any file systems, including `root`, `swap`, and `/usr`. Disk striping increases performance by spreading requests over multiple components. A combination of concatenation and striping increases capacity by grouping components into a single large logical device. Administration is simplified by the automatic replacement of failed components within a mirror and the dynamic growth of metadevices and file systems. UNIX file system (UFS) logging speeds up local directory operations and reboots and decreases the number of synchronous disk writes.

This chapter provides a high level overview of the features included with the DiskSuite software package. Use the following table to locate specific information.

3

Elements of the Metadisk Driver

The metadisk driver is implemented as a set of loadable, pseudo device drivers. The metadisk driver uses other physical device drivers to pass I/O requests to and from the underlying devices.

The metadisk driver resides between the file system interface and the device driver interface and interprets information from both above and below. After passing through the metadisk driver, information is received in the expected form by both the file system and by the device drivers. The metadisk is a loadable device driver and has all the same interfaces as any other device driver.

Figure 3-1 illustrates the position of a metadevice driver in the kernel hierarchy.

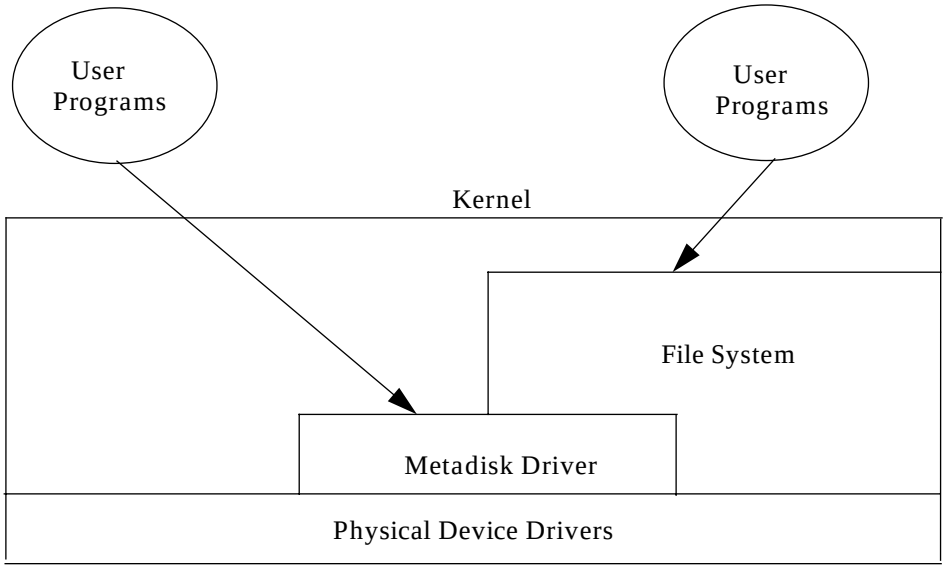


Figure 3-1 Location of the metadisk driver in the kernel hierarchy

An overview of the primary elements of the metadisk driver is provided in the following sections. These elements include:

- Metadevices
- Concatenation and striping
- Mirroring (metamirrors and submirrors)
- UFS logging
- Hot spares
- Disksets
- RAID devices

Metadevices

Metadevices are the basic functional unit of the metadisk driver. After you create metadevices, you can use them like physical disk partitions. These logical devices can be made up of one or more component partitions. You can configure the component partitions to use a single device, a concatenation of stripes, or stripe of devices.

Metadevices can provide increased capacity, higher availability, and better performance. To gain increased capacity, you create metadevices that are either concatenations or stripes. Mirroring and UFS logging provide higher availability and striping can help performance.

Metadevices are transparent to applications software and to component and controller hardware.

The standard metadevice name begins with "**d**" and is followed by a number. By default, there are 128 unique metadevices in the range 0-to-127. Additional metadevices can be added. Metadevice names are located in `/dev/md/dsk` and `/dev/md/rdsk`.

When using DiskSuite commands such as `metastat(1M)` and `metainit(1M)` or when setting up metadevices in the `md.tab` file, the metadevice name does not need to be fully qualified. For example, you can enter `d1` rather than `/dev/md/dsk/d1`. The examples presented in this manual use the short and long forms for metadevice names interchangeably.

Metadevices can be configured from IPI and SCSI devices on all SPARC systems and on SCSI and IDE devices on all x86 systems.

Concatenation and Striping

Each metadevice is either a concatenation or a stripe of component partitions. Concatenations and stripes work much the way the `cat`(1) program is used to concatenate two or more files together to create one larger file. When partitions are concatenated the addressing of the component blocks is done on the components sequentially. The file system can use the entire concatenation.

You can use a concatenated or striped metadevice for any file system with the exceptions of `root`, `swap`, `/usr`, `/var`, `/opt`, or any file system accessed during a Solaris upgrade or install.

Figure 3-2 illustrates the concatenation of three 327 Mbyte components. The block size is 512 bytes. The logical block address ranges are listed below the drives. The physical block address range for each of the three drives would be 0 to 669695. When concatenated, the logical block address range is sequential from 0-to-2009087. In this illustration the disk labels are not part of the logical block addresses.

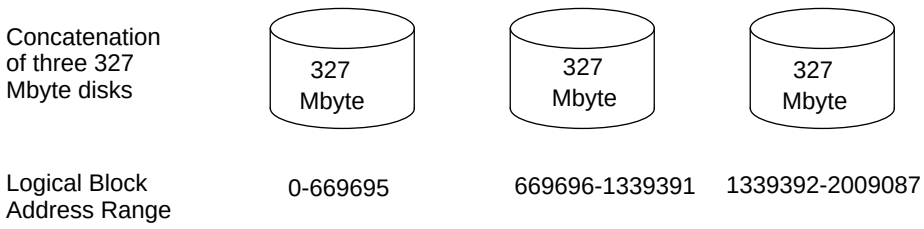


Figure 3-2 Concatenation of three 327-Mbyte drives

Striping is similar to concatenation, except the addressing of the metadevice blocks is interlaced on the components, rather than addressed sequentially. When stripes are defined, an interlace size is specified following the `-i` option. The interlace size is a number (for example, 8, 16, 32, etc.) followed by "k" for kilobytes, "m" for megabytes, or "b" for (512-byte) blocks. The units can be

specified in either uppercase or lowercase. If the size is not specified, it defaults to 16Kbytes. The interlace value tells DiskSuite how much data is placed on a component before moving to the next component of the stripe.

Because data is spread across a stripe, you gain increased performance as reads and writes are spread across multiple disk arms. Also, concurrent I/O requests may use different disk arms. This may be true of concatenation as well.

In Figure 3-3, three 327 Mbyte components are used to illustrate a stripe of component partitions. The block size is 512 bytes. The interlace value is 8 Kbytes (or 16 512-byte blocks). The same logical address range as shown for the concatenation in Figure 3-2 (0 to 2009087) applies to a stripe of the same component configuration. However, in striping, the logical block addresses on each component are alternated according to the interlace size specified. In this illustration the disk labels are not part of the logical block addresses.

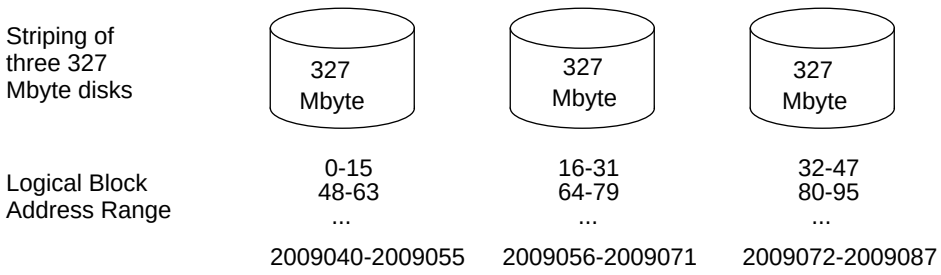


Figure 3-3 Striping of three 327-Mbyte drives with an interlace of 8 Kbytes

Figure 3-3 further illustrates how improved performance is gained through striping. For example, if a one Mbyte request were issued to this configuration, the data would be spread across the three components and the component arms on all components would be used to retrieve the data concurrently.

Mirroring

DiskSuite supports mirroring to as many as three separate metadevices. This enables the system to tolerate single-component failures with two-way mirroring and double failures with three-way mirroring. Mirroring can also be used for online backups of file systems.

To set up mirroring, you create a metamirror. A metamirror is a special type of metadevice made up of one or more other metadevices. Each metadevice within a metamirror is called a submirror.

Metamirrors can be given names such as `d0`. The same naming convention used for metamirrors is used for metadevices. For example, a metamirror could have the name `d1` and a metadevice might have the name `d2`, or visa versa.

After you define a metamirror (with the `metainit` command for example), you can add additional submirrors at a later date without bringing the system down or disrupting writes and reads to existing metamirrors. Submirrors are added with the `metattach(1M)` command, which attaches a submirror to the metamirror.

When the submirror is attached, all the data from another submirror in the metamirror is automatically written to the newly attached submirror. This is called resyncing. After the resyncing is complete, the new submirror is readable and writable. Once a `metattach` is performed, the submirrors remain attached (even when the system is rebooted) until they are explicitly detached with the `metadetach(1M)` command.

You can use other DiskSuite utilities to perform maintenance on metamirrors and submirrors. If a controller fails, any submirrors on that controller can be taken offline with the `metaoffline(1M)` utility. While a submirror is offline, DiskSuite keeps track of all writes to the metamirror. When the submirror is brought back online with the `metaonline(1M)` command, only the portions of the metamirror that were written (called *resync regions*) are resynced.

UNIX File System Logging

The UNIX file system (UFS) logging facility included with DiskSuite provides faster local directory operations, speeds up reboots, and decreases synchronous disk writes by safely recording file system updates in a log before they are applied to the UFS file system.

Note – The UFS logging facility can only be used with Solaris 2.4 or later releases.

UFS is the standard Solaris file system. UFS file systems are created when Solaris is installed or by users with the `newfs(1M)` command.

Because a system crash can interrupt system calls that are already in progress and thereby introduce inconsistencies, UFS file systems should be checked before they are mounted again. Mounting a UFS file system without first checking it and repairing any inconsistencies can cause panics or data corruption. Checking large file systems is a slow operation because it requires reading and verifying the file system data. With the UFS logging facility, UFS file systems do not have to be checked at boot time because the changes from unfinished system calls are discarded.

A pseudo device, called the *metatrans device*, is responsible for managing the contents of the log of file system updates. Like other metadevices, the metatrans device behaves the same as an ordinary disk device. The metatrans device is made up of two subdevices: the *logging device* and the *master device*. These can be disk partitions, metadevices, and metamirrors; but not metatrans devices.

Figure 3-4 illustrates the metatrans device `/dev/md/dsk/d1` and the two subdevices of which it's comprised. `/dev/dsk/c0t0d0s3` is the master device and `/dev/dsk/c1t0d0s3` is the logging device.

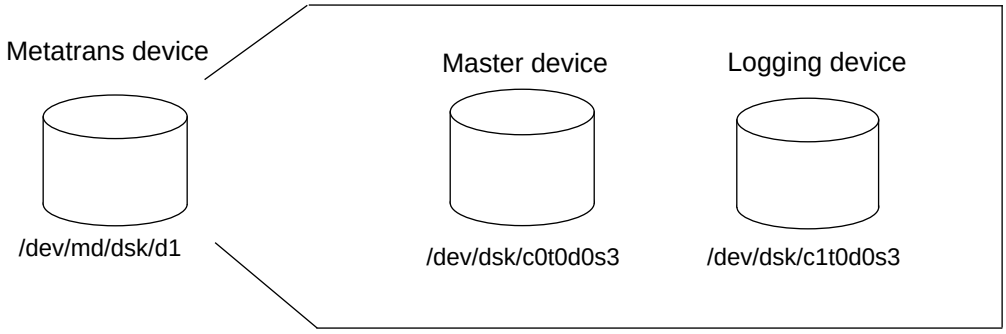


Figure 3-4 UFS Logging

As shown in Figure 3-4, the same naming convention used for metamirrors and metadevices is used for metatrans devices. For example, a metatrans device could have the name `d1` and a metamirror might have the name `d2`, or visa versa.

The logging device contains the log of file system updates. This log consists of a sequence of records, each of which describes a change to a file system. The master device contains an existing or a newly created UFS file system.

The master device can contain an existing UFS file system because creating a metatrans device does not alter the master device. The difference is that updates to the file system are written to the log before being “rolled forward” to the UFS file system. Likewise, clearing a metatrans device leaves the UFS file system on the master device intact.

As illustrated in Figure 3-5, a logging device can also be shared among metatrans devices. The figure shows the first metatrans device `/dev/md/dsk/d1` and the two subdevices of which it’s comprised. `/dev/dsk/c0t0d0s3` is the master device and `/dev/dsk/c1t0d0s3` is the logging device. The second metatrans device `/dev/md/dsk/d2` is shown with its own master device `/dev/dsk/c2t0d0s3` and the logging device `/dev/dsk/c1t0d0s3` it shares with the first metatrans device.

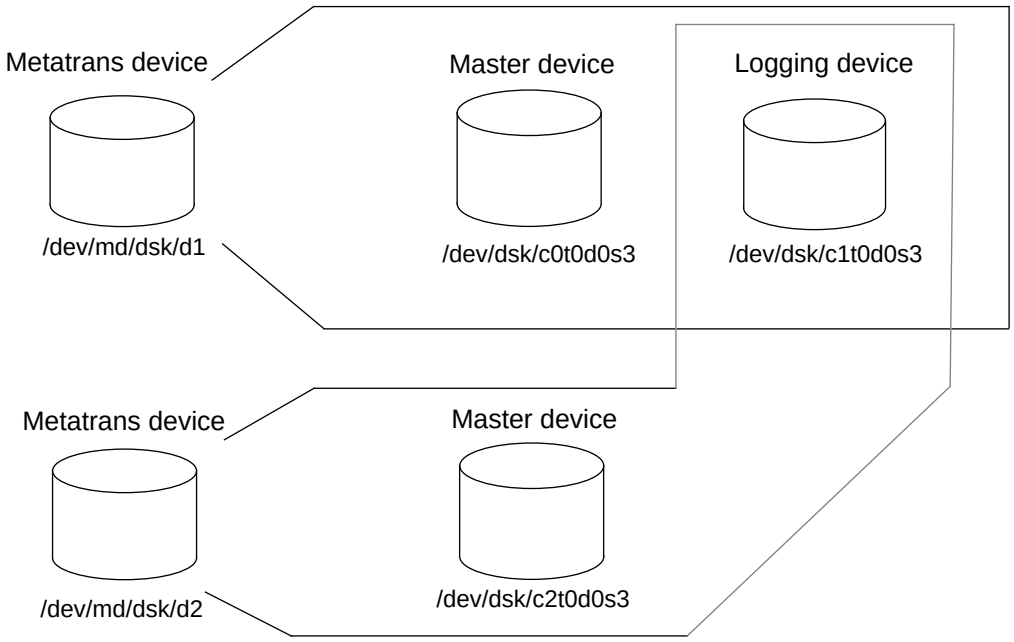


Figure 3-5 Shared logging device

Hot Spares

DiskSuite's hot spare facility automatically replaces failed submirror or RAID components, provided that a spare component is available and reserved. Hot spares are temporary fixes, used until failed components are either repaired or replaced. Hot spares provide further security from downtime due to hardware failures.

The analogy that best describes hot spares is spare tires for cars. However, when a component fails, you do not have to stop and change to the hot spare component manually. This occurs automatically and without interruption of service. When a component fails, the replicated data is simply copied from any of the other submirrors or regenerated from the other components in a RAID device. Writes continue to the other components of the submirror or RAID device containing the failed component.

Hot spares are defined within hot spare pools, which can be a shared resource for all the submirrors and RAID devices you have configured. Individual hot spares can be included in one or more hot spare pools. For example, you may have three submirrors and three hot spares. The three hot spares can be arranged as three hot spare pools, with each pool having the three hot spares in a different order of preference. This enables you to specify which hot spare is used first. It also improves availability by having more hot spares available.

Disksets

DiskSuite's diskset feature lets you set up groups of host machines and disk drives in which all of the hosts in the set are connected to all the drives in the set. A diskset is a grouping of two hosts and disk drives in which all the drives are accessible by both hosts. DiskSuite requires that the device name be identical on each host in the diskset. There is one metadevice state database per shared diskset, as well as the one on the local diskset.

RAID Devices

DiskSuite's RAID feature provides support for RAID devices. RAID stands for "Redundant Arrays of Inexpensive Disks." DiskSuite RAID devices support RAID Level 5.

RAID devices are comprised of three or more physical partitions. Each partition is referred to as a column. A RAID metadevice can be grown by concatenating additional partitions to the metadevice.

RAID level 5 uses multiple physical partitions used to simulate a single large slice (partition). A single sector on one of these physical slices contain either a sector's worth of contiguous data, or parity information relating to the data on the same sector of all other slices in the array.

In order to eliminate a parity partition as a bottleneck, no one physical partition will hold all of the parity information; it will be placed on different partitions for different sectors.

The advantages of a RAID Level 5 configuration are that it can recover from a single disk failure and that it can be more cost effective than mirroring disks.

State Database Replicas

State database replicas provide the non-volatile storage necessary to keep track of configuration and status information for all metadevices, metamirrors, metatrans devices, hot spares, and RAID devices. The replicas also keep track of error conditions that have occurred.

After a metadevice is configured, it is necessary for the metadevice driver to remember its configuration and status information. The metadevice state database is the metadevice driver's long term memory. The metadevice driver stores all the metadevice configuration information in the state database. This includes the configuration information about mirrors, submirrors, concatenations, stripes, metatrans devices, and hot spares.

If the replicated metadevice state database were to be lost, the metadevice driver would have no way of knowing any configuration information. This could result in the loss of all data stored on metadevices. To protect against losing the metadevice state database because of hardware failures, multiple replicas (copies) of the state database are kept.

These multiple replicas also protect the state database against corruption that can result from a system crash. Each replica of the state database contains a checksum. When the state database is updated, each replica is modified one at a time. If a crash occurs while the database is being updated, only one of the

replicas will be corrupted. When the system reboots, the metadevice driver uses the checksum embedded in the replicas to determine if a replica has been corrupted. Any replicas that have been corrupted are ignored.

If a disk that contains the metadevice state database is turned off, the metadevices remain fully functional because the database is retrieved from one of the replicas still in operation. Changes made to the configuration following the reboot are stored only in the replicas that are in operation when the system comes back up. If the disk drive that was turned off is later turned back on, the data contained in the replica stored on that disk is ignored. This is accomplished by comparing it with other replicas.

Expanding Mounted File Systems

You can expand mounted or unmounted UFS file systems with the DiskSuite concatenation facilities and the `growfs(1M)` command. The expansion can be performed without bringing down the system or performing a backup.

Mounted or unmounted file systems can be expanded up to the new size of the metadevice on which the file system resides.

Note – Once you have expanded a file system, it cannot be shrunk.

DiskSuite Commands and Utilities

There are several new utilities included with the DiskSuite package. An overview of the DiskSuite utilities follows. For a complete definition of the functionality and options associated with the utilities, refer to Appendix D.

- `metaclear(1M)` - Clears (deletes) all (or only specified) metadevices and/or hot spare pools from the configuration. After a metadevice is cleared, it must be reconfigured again with the `metainit` utility. `metaclear` does not clear metadevices that are currently in use (open). You can never clear `root` and `swap` metadevices.
- `metadb(1M)` - Reserves or releases space for the metadevice state databases, which are used in the event of a system failure to determine the status and configuration of the metadevices. All metadevice state databases contain

identical information, which guards against the loss of configuration information. If the status and configuration information is lost, the metadevices will no longer operate.

- `metadetach(1M)` - Detaches a submirror from a metamirror. After a submirror is detached, reads and writes to the detached metadevice are no longer performed. The command does not allow detaching the last remaining submirror.

  `metadetach` detaches a logging device from a metatrans device. Logging is disabled while the metatrans device lacks a logging device.

- `metahs(1M)` - The management utility for hot spares and hot spare pools.
- `metainit(1M)` - Configures the metadevices and hot spares according to the configuration specified either on the command line or in the `md.tab` file. This is the first command run before using metadevices.
- `metaoffline(1M)` - Prevents DiskSuite from reading and writing the offlined submirror. While the submirror is offline, all writes to the metamirror are recorded and are written when the submirror is brought back online. This command is only used on submirrors.
- `metaonline(1M)` - Resumes accesses to a submirror. When the command is specified, a resync is automatically invoked to resync only the regions written to while the submirror was offline. This command is used only on a submirror that has been taken offline by `metaoffline`.
- `metaparam(1M)` - Modifies parameters of metadevices and metamirrors. The parameters that can be modified are those displayed by either the `metaparam` or the `metastat` command. The interlace value of a striped metadevice can not be changed by `metaparam`.
- `metareplace(1M)` - Replaces a component of a submirror or RAID device with a new component. The utility automatically begins a resync operation to the new component.
- `metaroot(1M)` - Edits the system files, `/etc/vfstab` and `/etc/system`, so the system can be booted with the root file system on a metadevice.
- `metaset(1M)` - Administers sets of disks (*disksets*) shared for exclusive access between hosts.
- `metastat(1M)` - Reports the current status for the active metadevice(s) and hot spare pools that are specified. If a metadevice is not specified, the status of all metadevices and hot spare pools is reported.

- `metasync(1M)` - Performs mirror resync operation on submirrors in a metamirror or components in a RAID device. Applications have access to the metamirror or RAID device while the resyncs are in progress in the background. You will rarely need to run this command directly, as it is invoked at boot time.
- `metattach(1M)` - Attaches a metadevice to a metamirror as a new submirror. The utility automatically begins a resync operation to the new submirror. It can also be used to concatenate a new component to an existing metadevice without interrupting service.

  `metattach` attaches a logging device to a metatrans device.

One other utility associated with DiskSuite is:

- `growfs(1M)` - Nondestructively expands a mounted or unmounted file system up to the size of the physical device allocated for the file system.

System Files Associated with DiskSuite

There are three system files associated with DiskSuite that are used by the various utilities:

- `md.tab` - Used by the `metainit` and `metadb` commands as a workspace file. Each metadevice may have a unique entry in this file. Tabs, spaces, comments (using the pound sign (#) character), and continuation of lines (using the backslash (\) character) can be used in the file.

Note – The `md.tab` file is used only when creating metadevices, hot spares, or database replicas. This file is *not* automatically updated by the DiskSuite utilities. This file may have little or no correspondence with actual metadevices, hot spares, or replicas.

- `md.cf` - Automatically updated whenever the configuration is changed by the user. This is basically a disaster recovery file and should never be edited by the user. This file should never be used blindly after a disaster as the `md.tab` file. Be sure to carefully examine the file first.

Note – The `md.cf` file does not get updated when hot sparing occurs.

- `mddb.cf` - Created whenever the `metadb` command is run and is used by `metainit` to find the locations of the metadevice state database. You should never edit this file. Each metadevice state database replica has a unique entry in this file. Each entry contains the driver name and minor unit numbers associated with the block physical device where the replica is stored. Each entry also contains the block number of the master block, which contains a list of all other blocks in the replica.

Concatenating and Striping 4

This chapter discusses the specifics of the DiskSuite facilities for creating metadevices consisting of either concatenated or striped disk partitions. Refer to Chapter 3, “Overview of Solstice DiskSuite,” for overview information on concatenations and stripes.

Use the following table to locate specific information in this chapter.

Using Concatenations and Stripes

Concatenating disk partitions enables you to create a single metadevice with large capacity. This provides a solution for the space limitation a single partition poses. The logical disk block addresses are allocated sequentially to concatenated disk partitions.

You can use a concatenated or striped metadevice for any file system with the exception of `root`, `swap`, `/usr`, `/var`, `/opt`, or any other file system accessed during a Solaris upgrade or installation.

Striping performs the same function as concatenation, enabling you to create a single metadevice with large capacity. But striping differs from concatenation because the logical disk block addresses are allocated in an interlaced fashion, which can provide improved performance.

The interlace value for striping is user defined. The interlace size is a number (for example, 8, 16, 32) followed by either `k` for kilobytes, `m` for megabytes, or `b` for disk blocks. The suffix can be either uppercase or lowercase. If the interlace size is not specified, the size defaults to 16 Kbytes. A performance gain occurs when the I/O request is larger than the interlace size, because more disk arms are used to retrieve data or multiple requests spread over more disk arms.

DiskSuite supports concatenations consisting of a single component partition. This is supported to allow temporarily defining single component configuration in anticipation of adding more components in the future. Another use of a single component configuration is to enable mirroring of that component. Additional information on mirroring is provided in Chapter 5, “Mirroring.”

After the metadevice is configured, it can be used just as if it were a physical partition. As with physical partitions, a file system can be created on the metadevice. Most UNIX disk utilities will work normally on metadevices, with the exception of `format(1M)`.

In addition, all usual file system operations can be performed on a metadevice. The following list offers examples of file system operations:

- `mount(1M)` the metadevice on a directory
- `umount(1M)` a mounted metadevice
- Copy files to the metadevice
- Read and write files from and to the metadevice
- `ufsdump(1M)` and `ufsrestore(1M)` the metadevice

Defining Metadevice Configurations

There are two methods of defining either a concatenated or striped metadevice:

- Edit the `md.tab` file to define the concatenation or stripe, then run the `metainit(1M)`command; or
- Include the definition of the concatenation or stripe on the command line with the `metainit(1M)`command.

The information that you include either on the command line or in `md.tab` will differ depending on whether the configuration is a concatenation or a stripe. For a complete discussion of the `md.tab` file, refer to Appendix A, “Solstice DiskSuite Files.”

A drawback of concatenations, stripes, and concatenated stripes is that the loss of a single physical partition can cause the loss of service to the entire metadevice. This can be solved through mirroring and the use of hot spares. These two topics are discussed in subsequent chapters.

The following sections show how to define concatenations, stripes, and concatenated stripes. Additional step-by-step examples are given in this chapter that show the entire procedure for setting up metadevices.

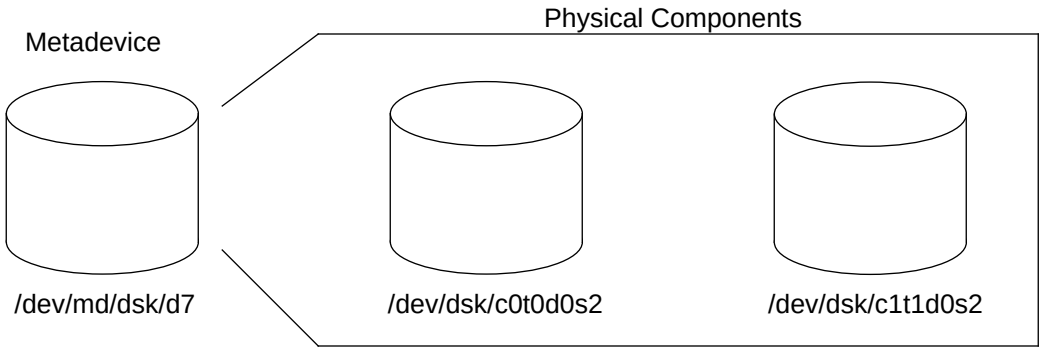
Concatenated Metadevices

Concatenation can help solve the single partition capacity limitation. For example, a single database could exist on a single metadevice spanning multiple physical partitions.

Figure 4-1 shows two physical components and a metadevice. The physical components are `/dev/dsk/c0t0d0s2` and `/dev/dsk/c1t1d0s2` and the metadevice is `/dev/md/dsk/d7`. As illustrated in Figure 4-1, a metadevice is a ‘‘virtual disk’’ and can be used like a physical disk partition.

Following the illustration are examples of the information that would be required, either on the command line or in your `md.tab` file, to define a concatenation of the configuration shown.

Figure 4-1 Concatenated Metadevice



To set up a concatenation of the two physical components shown in Figure 4-1, the following would be added to your `md.tab` file, or specified directly as arguments to `metainit` on the command line.

```
d7 2 1 /dev/dsk/c0t0d0s2 1 /dev/dsk/c1t1d0s2
```

In the above entry, the number `2` represents the two stripes of devices that are being concatenated together to create the metadevice. In this case, each stripe is made up of one component, thus the number `1` must be displayed in front of each component.

Striped Metadevices

When components are configured into stripes, faster I/O throughput can be obtained. The ideal configuration of stripes would have each component on a separate disk controller. This configuration would deliver the best performance.

A configuration that delivers good performance would have the components on separate disks, but perhaps on the same controller.

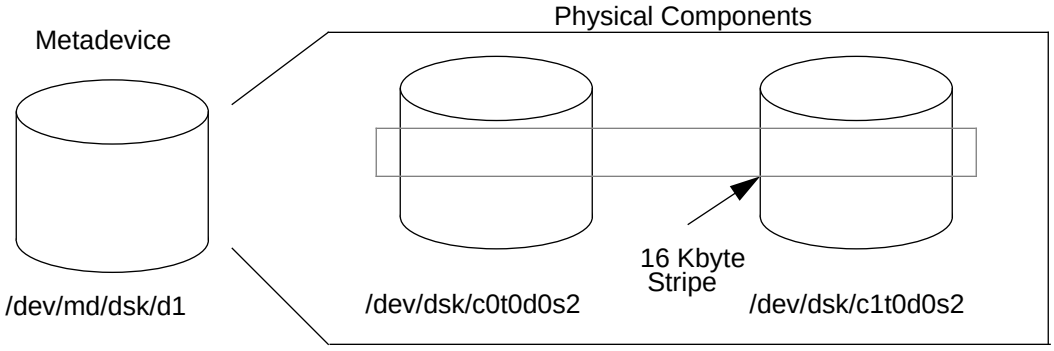
Note – Striping the partitions on a single disk should be avoided. This will hurt performance.

Changing the interlace values of an existing stripe is not supported. In order to change the value, the metadevice must be backed up, redefined with a different value, and restored.

Figure 4-2 shows two physical components and a metadevice. The physical components are `/dev/dsk/c0t0d0s2` and `/dev/dsk/c1t0d0s2` and the metadevice is `d1`. The 16 Kbyte stripe defined in this example is shown as a dotted rectangle.

Following the illustration are examples of the information that would be required, either in the `/etc/opt/SUNWmd/md.tab` file or on the command line as arguments to `metainit`, to define a stripe of the configuration shown.

Figure 4-2 Striped Metadevice



To create a stripe of the two physical components shown in Figure 4-2, enter the following as arguments to the `metainit` command on the command line, or add the following to your `/etc/opt/SUNWmd/md.tab` file.

```
d1 1 2 /dev/dsk/c0t0d0s2 /dev/dsk/c1t0d0s2 -i 16k
```

In the above entry, the number `1` represents the one row of devices that is being striped to create the metadevice. In this case, the row is made up of the two components, thus the number `2` must be displayed in front of the two components. The optional `-i 16k` sets the striped interlace value at 16 kilobytes. If the `-i` and a value are left off, the interlace size defaults to 16 Kbytes.

Metadevices Defined as Concatenated Stripes

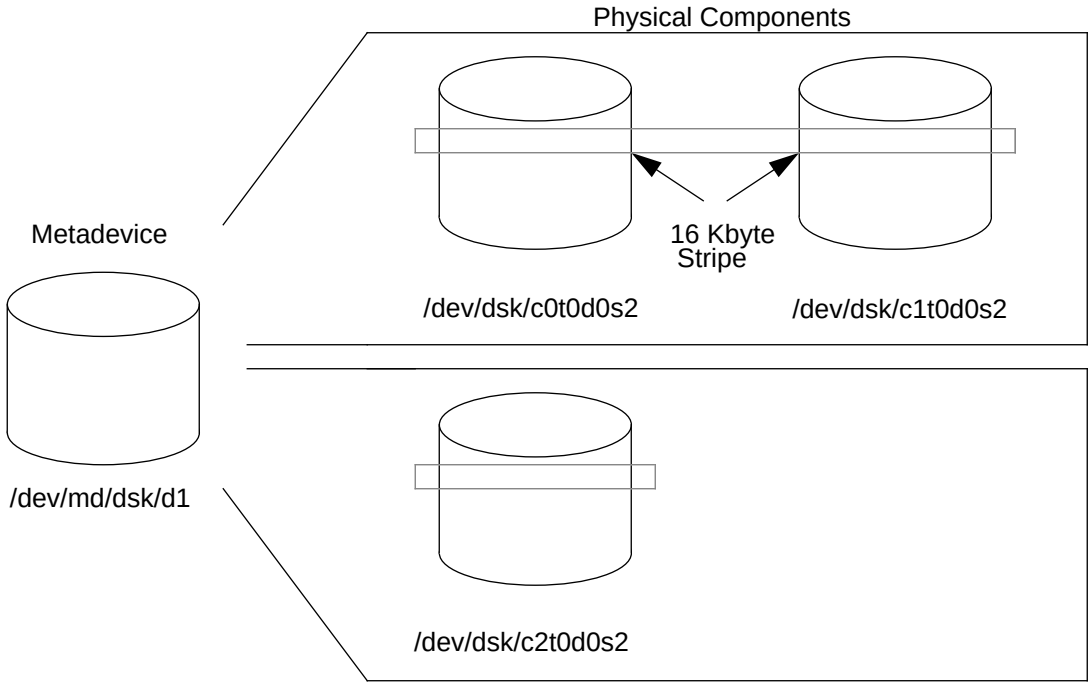
Concatenated stripes enable you to expand an existing stripe and provide some improved performance over simple concatenations.

Figure 4-3 shows three physical components and a metadevice. The physical components are `/dev/dsk/c0t0d0s2`, `/dev/dsk/c1t0d0s2`, and `/dev/dsk/c2t0d0s2`. The metadevice is `d1`.

In Figure 4-3, the three physical components make up a metadevice that is a concatenation of two stripes. The first stripe (shown as a dotted rectangle) is made up of two physical components and the second stripe is one component.

Following the illustration is an example of the information that would be required, either on the command line or in your `/etc/opt/SUNWmd/md.tab` file, to define the concatenation of stripes shown.

Figure 4-3 Concatenation of Two Stripes Into a Metadevice



To create a concatenation of two stripes of the three physical components shown in Figure 4-3, add the following to your `md.tab` file, or specify the following on the command line as arguments to `metainit`.

```
d1 2 2 /dev/dsk/c0t0d0s2 /dev/dsk/c1t0d0s2 -i 16k \
     1 /dev/dsk/c2t0d0s2
```

In the above entry, the first number `2` represents the two rows of devices that are being concatenated to create the metadevice. The first row is made up of `/dev/dsk/c0t0d0s2` and `/dev/dsk/c1t0d0s2`, thus the second number `2` is in front of these component names. The second row is made up of `/dev/dsk/c2t0d0s2`, which is preceded by the number `1`. The optional `-i 16k` sets the striped interlace value at 16 kilobytes for the first row. Because there is only one component in the second row, there is no interlace value.

If there were more than one component, they would have the same interlace value as the previous row, unless a different value is specified.

Replacing Failed Components

A situation at your site could arise that would require removing a component from a concatenation or a stripe (for example, if a component is reporting soft errors). There are two solutions available for replacing a component:

1. If you already have the metadevice mirrored (described in Chapter 5, “Mirroring”), the faulty component can be replaced by running the `metareplace(1M)` command. `metareplace` automatically starts a resync on the new component.
2. If you do not have the metadevice mirrored, use the following basic procedure to replace the failing component (if another disk is already attached to the system):
   a. Unmount the file system (`umount`), if there is one on the metadevice.
   b. Run the `ufsdump` command on the metadevice.
   c. Run the `metaclear` command on the metadevice.

d. Edit the `md.tab` file to change the name of the failing component in the metadevice. (If the metadevice is a stripe, the new component must be the same size as the failed one. If the metadevice is a concatenation, the new component must have at least the same capacity as the failed component.)

e. Run `metainit` on the metadevice.

f. Run `newfs` on the metadevice.

g. Run `ufsrestore` to put the data back online.

Clearing Concatenations and Stripes

If you decide to clear (delete) a concatenation or stripe and use the components in a traditional fashion, the procedure requires very little effort.

Warning – Any data that is on the concatenation or stripes will be lost when the metadevice is cleared. The data should be backed up to tape or disk.

The procedure for clearing concatenations or stripes is described in the following example. In this example, the metadevice that is being cleared is named `d8`.

1. **Back up the data currently on the components that make up the metadevice (either the stripe or concatenation).**

2. **Stop access to the data on the metadevice.**
   For example, if a file system resides on the metadevice, unmount the file system.

```
# /sbin/umount /dev/md/dsk/d8
```

3. **Clear the metadevice, using the `metaclear` command.**
   For example:

```
# /usr/opt/SUNWmd/sbin/metaclear d8
```

4. **You may want to remove the definition from the `/etc/opt/SUNWmd/md.tab` file.**
   For example, you would remove the following line:

```
d8 3 1 /dev/dsk/c0t0d0s0 1 /dev/dsk/c0t1d0s0 1 /dev/dsk/c0t2d0s0
```

Hardware and Software Considerations

There are both hardware and software considerations that affect concatenations and stripes. The only software consideration involves the values assigned to the interlace size when building a stripe.

The hardware considerations include mixing different size components, number of controllers, mixing components with different geometry, and the I/O load on the bus.

Assigning Interlace Values

The key to the improved performance using striping is the interlace value. The value is user configurable at the time a metadevice is created. Thereafter, the value cannot be modified.

The interlace value defaults to 16 Kbytes. This is a reasonable value for most applications. In general, a smaller interlace value increases CPU time while a larger interlace value decreases CPU time.

Mixing Different Size Components

When different size disk components are used in a stripe, some disk space will be unused unless you assign the unused portion to another metadevice.

This is because the stripe is limited by the smallest partition in the configuration (*n* times the smallest component, where *n* is the number of components in the stripe). For example, if you have two 327-Mbyte partitions and one 661-Mbyte partition in a stripe the stripe, will only use 327 Mbytes of the space on the 661-Mbyte partition.

To assign the unused disk space to another metadevice, the component must be repartitioned (using `format(1M)`).

Using Components With Different Geometry

All components in a stripe or concatenation should have the same number of sectors and tracks per cylinder. This is referred to as the disk geometry. The geometry is related to the capacity of the drive. Disk geometry varies depending on the manufacturer and type.

The problem with the differing component geometries is that the UFS file system attempts to lay out file blocks in an efficient manner. UNIX counts on a knowledge of the component geometry and uses cylinder groups to attempt to minimize seek distances. If all components do not have the same geometry, the geometry of the first component is what is reported to the file system. This may cause the efficiency of the file system to suffer.

Controllers

Building a stripe with all the component partitions on the same controller will adversely affect I/O performance. Also, creating a stripe of components on different controller types can affect performance, because some controllers are faster than others. The I/O throughput will be limited by the slowest controller.

An example of a controller limiting performance occurs when several devices (for example, 3-Mbyte per second disks) are attached to the same controller. In this instance, the throughput may be limited to the throughput of the controller and not the sum of the devices.

Bus Load

The data traffic load on the VMEbus or SBus can subtract from the performance gains you receive with the addition of controllers. This can be especially noticeable when stripes are defined. For example, a configuration of four disk controllers all connected to the same VMEbus may not give the maximum speed increase (four times the performance). This is because the bandwidth of the VMEbus might limit the throughput.

An example of this would be a VMEbus system with a throughput of approximately 7 Mbytes per second. A configuration on this system that adds four IPI controllers with a throughput of 6 Mbytes per second each would not increase throughput performance with striping.

However, on another VMEbus system with throughput of 22 Mbytes per second, a striping configuration that includes additional controllers would increase performance.

High transfer rate components on the SBus might encounter similar limitations.

Examples

Examples of three types of configurations are provided in this section.

These examples include:

- Striping three components
- Concatenating eight components
- Concatenating stripes of components

Striping Three Components

The following example shows how to define a stripe of three components. The stripe will have an interlace value of 32 Kbytes.

In the following example, the three components being striped are named `/dev/dsk/c0t3d0s7`, `/dev/dsk/c0t1d0s7`, and `/dev/dsk/c0t2d0s7`. The striped metadevice is named `d8`.

1. **Edit the `/etc/opt/SUNWmd/md.tab` file, adding a line that defines the striped metadevice.**

```
d8 1 3 /dev/dsk/c0t3d0s7 /dev/dsk/c0t1d0s7 \
        /dev/dsk/c0t2d0s7 -i 32k
```

2. **Use `metainit -n` to verify that the information in the `/etc/opt/SUNWmd/md.tab` file is accurate.**
   The `-n` option enables you to check your entry.

```
# /usr/opt/SUNWmd/sbin/metainit -n d8
```

3. **If the configuration is accurate, run** `metainit` **to begin using the striped metadevice.**

```
# /usr/opt/SUNWmd/sbin/metainit d8
```

Concatenating Eight Components

The following example shows how to define a concatenation of eight components.

In the following example, the eight components being concatenated are named `/dev/dsk/c0t0d0s2`, `/dev/dsk/c0t1d0s2`, `/dev/dsk/c0t2d0s2`, `/dev/dsk/c0t3d0s2`, `/dev/dsk/c2t0d0s2`, `/dev/dsk/c2t1d0s2`, `/dev/dsk/c2t2d0s2`, and `/dev/dsk/c2t3d0s2`. The metadevice being defined is named `d16`.

1. **Edit the** `/etc/opt/SUNWmd/md.tab` **file, adding the names of the components and the name of the concatenated metadevice.**

```
d16 8 1 /dev/dsk/c0t0d0s2 1 /dev/dsk/c0t1d0s2 \
      1 /dev/dsk/c0t2d0s2 1 /dev/dsk/c0t3d0s2 \
      1 /dev/dsk/c2t0d0s2 1 /dev/dsk/c2t1d0s2 \
      1 /dev/dsk/c2t2d0s2 1 /dev/dsk/c2t3d0s2
```

2. **Use** `metainit -n` **to verify that the information in the** `/etc/opt/SUNWmd/md.tab` **file is accurate.**
   Using the `-n` option enables you to check your entry.

```
# /usr/opt/SUNWmd/sbin/metainit -n d16
```

3. **If the configuration is accurate, run** `metainit` **to begin using the concatenated metadevice.**

```
# /usr/opt/SUNWmd/sbin/metainit d16
```

Concatenating Stripes of Components

The following example shows how to define a concatenation of two stripes. Each stripe consists of four 1-Gbyte components. The interlace value being assigned in this example is 16 Kbytes.

In the following example, the two sets of four components being striped are named `/dev/dsk/c0t0d0s7`, `/dev/dsk/c0t1d0s7`, `/dev/dsk/c0t2d0s7`, `/dev/dsk/c0t3d0s7`, `/dev/dsk/c2t0d0s2`, `/dev/dsk/c2t1d0s2`, `/dev/dsk/c2t2d0s2`, and `/dev/dsk/c2t3d0s2`. The striped metadevice being defined is named `d24`.

1. **Edit the `/etc/opt/SUNWmd/md.tab` file, adding the names of the components and the name of the metadevice.**

```
d24 2 4 /dev/dsk/c0t0d0s7 /dev/dsk/c0t1d0s7 \
        /dev/dsk/c0t2d0s7 /dev/dsk/c0t3d0s7 -i 16k \
      4 /dev/dsk/c2t0d0s2 /dev/dsk/c2t1d0s2 \
        /dev/dsk/c2t2d0s2 /dev/dsk/c2t3d0s2
```

2. **Use `metainit -n` to verify that the information in the `/etc/opt/SUNWmd/md.tab` file is accurate.**
   The `-n` option enables you to check your entry.

```
# /usr/opt/SUNWmd/sbin/metainit -n d24
```

3. **If the configuration is accurate, run `metainit` to begin using the concatenated metadevice.**

```
# /usr/opt/SUNWmd/sbin/metainit d24
```

4

Mirroring 5

DiskSuite provides the ability to replicate data stored on a particular metadevice onto as many as two additional (for a total of three) metadevices, referred to as submirrors. By setting up a minimum configuration of a two-way mirror you can recover from a single-component failure and perform online backups of file systems.

Mirroring components improves data reliability. An error on a component does not cause the entire mirror to fail. For further reliability, DiskSuite provides a facility for creating hot spare components. For further information on this utility, see Chapter 7, “Hot Spares.”

Although mirroring helps ensure data reliability, the I/O performance can suffer on some systems. However, the performance penalty can be minimized by the correct configuration of the mirrors.

To get maximum protection and performance, place mirrored metadevices on different physical components (disks) and on different disk controllers. Since the primary purpose of mirroring is to maintain availability of data, defining mirrored metadevices on the same disk is not recommended. If the disk were to fail, both metadevices would fail.

5

This chapter provides information and procedures for mirroring components using the DiskSuite software. Use the following table to locate information specific to your task.

Operation of Mirrors

Mirroring is accomplished by the creation of metamirrors. A metamirror is a special type of metadevice that is made up of one or more other metadevices (concatenations or stripes). Each metadevice within a metamirror is called a submirror.

Mirrors or metamirrors use the same naming convention used for other metadevices. After a metamirror is defined, additional submirrors can be added at any time without disruption of writes or reads to the existing metamirror.

Metamirrors should be defined with a single submirror using the `metainit` command. Additional submirrors must be added with the `metattach` command. Metamirrors can also be defined in the `/etc/opt/SUNWmd/md.tab` file, but you must run `metainit` to activate them. When you are mirroring existing file systems or data, be sure that the existing data is contained on the submirror initially defined with the metamirror. When a second submirror is subsequently attached, data from the initial submirror is copied onto the attached submirror.

In general, it is best to avoid creating multi-way metamirrors using the `metainit` command *without* attaching the other submirrors. For example, the following use of the `metainit` command is not recommended:

```
# metainit d0 -m d1 d2
```

When you use `metainit` in this manner, the following message is displayed:

```
WARNING: This form of metainit is not recommended.
         The submirrors may not have the same data.
      Please see ERRORS in metainit(1M) for additional
information.
```

When the `metattach` command is not used to attach the additional submirrors, no resync operation occurs.

Caution – In cases where you are mirroring existing data, be sure that the initial submirror contains the data.

One-way metamirrors (containing a single submirror), can be defined either in the `/etc/opt/SUNWmd/md.tab` file, or with the metainit command directly from the command line. For convenience, it is recommended that all simple metadevices be defined using the `/etc/opt/SUNWmd/md.tab` file. This will make manual operations (attaching, detaching, and replacing metadevices) easier to deal with.

There are basically two ways in which you can set up metamirrors:

- By defining one-way metamirrors in the `/etc/opt/SUNWmd/md.tab` file, running the `metainit` command with the `-a` option, and then performing subsequent instances of the `metattach` command (or use a shell script) to attach the submirrors.
- By defining one-way mirrors using the `metainit` command either manually or from a shell script, and then performing subsequent additions of submirrors using the `metattach` command.

The examples in this chapter show metadevices and one-way metamirrors defined in the `/etc/opt/SUNWmd/md.tab` file. Subsequent submirror additions are shown separately below the `md.tab` file definitions.

Note – Remember that definitions in the `/etc/opt/SUNWmd/md.tab` file do not become effective until they are activated using the `metainit` command.

An example is a metamirror that is made up of four physical components and two submirrors (metadevices). The physical components are named `/dev/dsk/c0t0d0s0`, `/dev/dsk/c1t0d0s0`, `/dev/dsk/c2t0d0s0`, and `/dev/dsk/c3t0d0s0`. The two submirrors are named `/dev/md/dsk/d1` and `/dev/md/dsk/d2`. The metamirror is named `/dev/md/dsk/d50.`

To set up this metamirror using the `/etc/opt/SUNWmd/md.tab` file, you would add the following:

```
#
# mirror concatenation of two components
#
/dev/md/dsk/d50 -m /dev/md/dsk/d1
/dev/md/dsk/d1 2 1 /dev/dsk/c0t0d0s0 1 /dev/dsk/c1t0d0s0
/dev/md/dsk/d2 2 1 /dev/dsk/c2t0d0s0 1 /dev/dsk/c3t0d0s0
```

In the above `/etc/opt/SUNWmd/md.tab` entry, the `-m` specifies the one-way mirror consists of the metadevice, `/dev/md/dsk/d1`. The two metadevices to be used as submirrors are then defined in the two lines that follow. (The metadevices are concatenations, as explained in Chapter 4, “Concatenating and Striping.”)

To activate the metamirror you’ve defined, you would use the `metainit` command with the `-a` option as follows::

```
# metainit -a
```

The second metadevice is then attached with the `metattach` command as follows:

```
# metattach d50 d2
```

The metadevice, `/dev/md/dsk/d2` is now a submirror for the metamirror `/dev/md/dsk/d50`.

Defining Metamirrors

All metamirrors that have been defined in the `/etc/opt/SUNWmd/md.tab` file can be configured at the same time or they can be configured individually using the `metainit(1M)` command. `metainit` configures the metadevices according to the configurations specified either on the command line or in the `/etc/opt/SUNWmd/md.tab` file.

When used with the `-a` option, `metainit` configures all metadevices defined in the `/etc/opt/SUNWmd/md.tab` file. If `metainit -a` is used, all submirrors are configured before the metamirror, regardless of the order in which the entries are made in the `/etc/opt/SUNWmd/md.tab` file.

If `metainit` is used to configure a metamirror and any of the submirrors are not configured at the time, `metainit` fails. For example, assume the following entries are made in the `/etc/opt/SUNWmd/md.tab` file.

```
/dev/md/dsk/d25 -m /dev/md/dsk/d26
/dev/md/dsk/d26 1 1 /dev/dsk/c0t1d0s0
/dev/md/dsk/d27 1 1 /dev/dsk/c0t2d0s0
```

Running `metainit -a` would activate the submirrors `/dev/md/dsk/d26` and `/dev/md/dsk/d27` and then activate the one-way mirror `/dev/md/dsk/d25`. If the submirrors and metamirror are activated individually, they should be done as follows:

```
# /usr/etc/metainit /dev/md/dsk/d27
# /usr/etc/metainit /dev/md/dsk/d26
# /usr/etc/metainit /dev/md/dsk/d25
```

Next, you would use the `metattach` command to attach the submirror `/dev/md/dsk/d27` to the metamirror `/dev/md/dsk/d25` as follows:

```
# /usr/etc/metattach /dev/md/dsk/d25 /dev/md/dsk/d27
```

Metamirror Options

DiskSuite supports several options to optimize metamirror performance for needs at individual sites. These options deal with:

- Setting the order in which metamirrors are resynced during reboot
- Read policy for metamirrors
- Write policy for metamirrors

You define these options using the `metainit` command when you initially configure a metamirror, or using the `metaparam` command after the metamirror has been set up. For more information regarding using the `metaparam` command, see "Changing Metamirror and Submirror Options" on page 97 or the `metaparam(1M)` manpage.

Resync Options

A pass number in the range 0-9 at the end of an entry defining a metamirror determines the order in which that mirror is resynced during a system reboot. The default is 1. Smaller pass numbers are resynced first. If 0 is used, the resync is skipped. A 0 should only be used for metamirrors mounted as read only. If different metamirrors have the same pass number, they are resynced concurrently. The following example shows three metamirrors defined in a `/etc/opt/SUNWmd/md.tab` file.

```
/dev/md/dsk/d51 -m /dev/md/dsk/d52 2
/dev/md/dsk/d61 -m /dev/md/dsk/d62 2
/dev/md/dsk/d71 -m /dev/md/dsk/d72
```

In the above example, the metamirror `/dev/md/dsk/d71` is resynced first, then `/dev/md/dsk/d51` and `/dev/md/dsk/d61` are resynced concurrently.

Read Options

There are three kinds of read options associated with metamirrors. The default read option is "balanced load." The balanced load means all reads are made in a round-robin order from all the submirrors in the metamirror.

If a `-g` is specified following a metamirror entry in the `/etc/opt/SUNWmd/md.tab` file or following a command line metamirror entry with the `metainit` utility, the metadisk driver performs "geometric" reads. This option provides faster performance on sequential reads or when you're using disks with track buffering.

Geometric reads allow read operations to be divided among submirrors on the basis of a logical disk block address. For instance, with a three-way submirror the disk space on the metamirror is divided into three (equally sized) logical address ranges. Reads from the three regions are then performed by separate submirrors (for example, reads to the first region are performed by the first submirror).

If a `-r` option is specified, all reads are directed to the first submirror. This option cannot be used in conjunction with a `-g`.

The following example illustrates the use of the `-r` and `-g` options:

```
# metainit /dev/md/dsk/d75 -m /dev/md/dsk/d76 -g
# metattach /dev/md/dsk/d75 /dev/md/dsk/d77
# metainit /dev/md/dsk/d80 -m /dev/md/dsk/d81 -r
# metattach /dev/md/dsk/d80 /dev/md/dsk/d82
```

In the above example, reads from the `/dev/md/dsk/d75` metamirror will be performed geometrically from the two submirrors (`/dev/md/dsk/d76` and `/dev/md/dsk/d77`). When reads are made from the `/dev/md/dsk/d80` metamirror the submirror, `/dev/md/dsk/d81`, will be used.

Write Options

Writes to the submirror of a metamirror are either performed in parallel or serially. Parallel writes are the default action of the metadisk driver, meaning the writes are dispatched to all submirrors simultaneously. If a `-S` (uppercase 's') option is specified following a metamirror, the writes happen serially in the order in which the submirrors are specified in the metamirror. The following is an example of a metamirror defined with the `-S` option.

```
# metainit /dev/md/dsk/d90 -m /dev/md/dsk/d91 -S
# metattach /dev/md/dsk/d90 /dev/md/dsk/d92
# metattach /dev/md/dsk/d90 /dev/md/dsk/d93
```

In the above example, a write to `/dev/md/dsk/d90` will first write the data to `/dev/md/dsk/d91`. Once the write to `/dev/md/dsk/d91` is complete, a write to `/dev/md/dsk/d92` begins and after it finishes, the write to the third submirror, `/dev/md/dsk/d93`, will be made.

Resyncing Mirrors

If submirrors need to be replaced, attached, or taken offline for other reasons, the submirrors must be resynced. Resyncing is the act of copying data from one submirror to another and ensures the data on all submirrors is identical. DiskSuite offers two types of resyncing:

- Full resync
- Optimized resync

These resyncs both perform the basic function of copying data from a readable submirror to another submirror. While this copy takes place, the metamirror remains readable and writable by users.

A full resync must be performed when a new submirror is added to a metamirror and the user wants to bring it in sync with other submirrors that are part of that metamirror. Full resyncs are performed whenever the `metattach(1M)` command is used.

`metattach` attaches metadevices to metamirrors and resyncs the newly attached submirror to the mirror. The metadevices remain attached, even during system reboots, until the `metadetach` command is run.

If the system crashes while any resync is in progress, DiskSuite will finish the resync when the system comes back up.

An optimized resync occurs when the system boots following a crash or a reboot. Metamirror resyncs following a system crash are necessary because the crash may have interrupted a mirror write, thus some submirrors in the metamirror may be out of sync.

DiskSuite keeps track of regions of all mirrors. When a crash occurs, the metadisk driver knows which regions are not in sync. Subsequently, when the system reboots, only the regions that are not in sync are updated during this type of resync. For large metamirrors, this could be time consuming. This does not cause major problems during reboot, because system activity, including `fsck(1M)`, can continue. However, system throughput is degraded.

`metaonline(1M)` performs an optimized resync similar to the type of optimized resync that is normally performed at boot time. `metaonline` only resyncs the data that was written to the metamirror while that submirror was offline.

Checking the Status of Mirrors

The `metastat(1M)` command is used to display the current status for each metadevice or hot spare pool, or of specified metadevices or hot spare pools. By default, all information pertaining to metadevices and hot spare pools is displayed by `metastat`.

The `-p` option can be used with `metastat`. When you specify the `-p` option, the output format is similar to that of the `/etc/opt/SUNWmd/md.tab` file. This option is mainly for viewing the metadevive configuration only; without status information.

The name of a metadevice or hot spare pool can be specified on the command line as an argument to the `metastat` command. If one is specified, only the information pertaining to that metadevice or hot spare pool is displayed.

If no metadevice or hot spare pool is specified, `metastat` displays information for all active metadevices and hot spare pools. An example of the default output from `metastat` follows:

```
# /usr/opt/SUNWmd/sbin/metastat
d13: Mirror
    Submirror 0: d14
      State: Needs maintenance
    Submirror 1: d15
      State: Okay
    Submirror 2: d16
      State: Okay
    Pass: 1
    Read option: roundrobin (default)
    Write option: parallel (default)
    Size: 183750 blocks
      .
      .
      .
```

In the above output, information about the metamirror, `/dev/md/dsk/d13`, is displayed. `metastat` shows that the three submirrors, `/dev/md/dsk/d14`, `/dev/md/dsk/d15`, and `/dev/md/dsk/d16` are configured. The state of each submirror is also provided. There are three possible states: ‘‘Okay,’’ ‘‘Resyncing,’’ and ‘‘Needs maintenance.’’

The next four lines deal with options that are set for the mirror. The ‘‘Pass’’ number defines when the mirrored metadevices will be resynced when the system reboots. The ‘‘Read option’’ specifies round-robin reads (the default). The ‘‘Write option’’ specifies parallel writes (the default). See “Metamirror Options” on page 70 and Appendix A, “Solstice DiskSuite Files” for additional information on setting up the `/etc/opt/SUNWmd/md.tab` file.

The size information is the total number of blocks included in the metamirror.

Output from `metastat` continues below with a submirror display.

```
      .
      .
      .
d14: Submirror of d13
   State: Needs maintenance
   Invoke: metareplace /dev/md/dsk/d13 /dev/dsk/c0t3d0s0 \
           <new  device>
   Hot spare pool: hsp003
   Size: 183750 blocks
   Stripe 0:
     Device             Start Block    Dbase    State  Hot Spare
      c0t2d0s0                0          No        Okay
   Stripe 1: (Interlace = 16 blocks)
     Device             Start Block    Dbase    State  Hot Spare
      c0t3d0s0              630          Yes       Maintenance
      c0t4d0s0              630          Yes       Maintenance
   Stripe 2:
     Device             Start Block    Dbase    State  Hot Spare
      c0t0d0s0              0            No        Maintenance
      .
      .
      .
```

In the above output, information about the submirror is displayed. `metastat` shows the metamirror (`/dev/md/dsk/d13`) which contains the submirror `/dev/md/dsk/d14`. The state of the submirror is shown on the second line.

There are four possible states: ‘‘Okay,’’ ‘‘Resyncing,’’ ‘‘Maintenance,’’ and “Last Erred.” If ‘‘Maintenance’’ is displayed, the next line, ‘‘Invoke’’ specifies which command should be run to eliminate the problem. The interlace value is only shown if there are two or more devices in the stripe.

For example, to correct the maintenance required in the above example, the following command might be run:

```
# /usr/opt/SUNWmd/sbin/metareplace /dev/md/dsk/d13 \
        /dev/dsk/c0t3d0s0 /dev/dsk/c0t5d0s0
```

In this example, the device, `/dev/dsk/c0t5d0s0`, is used for the “`<new device>`.” It is possible that you may have to run `metastat` again to see if additional commands are needed to solve the ‘‘Needs maintenance’’ state.

The hot spare pool that has been assigned to the submirror is displayed on the following line. Next, the size of the submirror (in blocks) is displayed.

`metastat` also displays information about each component in the stripe, as shown with a separate entry. Each entry consists of the ‘‘Device,’’ ‘‘Start Block,’’ ‘‘Dbase,’’ ‘‘State,’’ and ‘‘Hot Spare.’’

The ‘‘Device’’ is the name of the device that is in the stripe.

The ‘‘State’’ is the state of the device, which is either ‘‘Okay,’’ ‘‘Resyncing,’’ ‘‘Maintenance,’’ or “Last Erred.” These states are defined as follows:

- “Okay” - The component is operating properly.
- “Resyncing” - The component is actively being resync’ed.
- “Maintenance” - The component has encountered an I/O error or an open error. All reads and writes to and from this component have been discontinued. See “Replacing and Enabling Submirror Components” on page 96 for information on component replacement.
- “Last Erred” - The component has encountered an I/O error or an open error, however, the data is not replicated elsewhere due to another component failure. I/O is still performed on the component. If I/O errors result, the mirror I/O will fail. See “Replacing and Enabling Submirror Components” on page 96 for information on component replacement.

The ‘‘Start Block’’ is the block on which the component begins.

The “Dbase” field specifies whether or not a database resides on the component.

The final field is “Hot Spare” which specifies the device name of the hot spare that has been used for a replacement when the device failed.

Output from `metastat` continues below with a metadevice display.

```
      .
      .
      .
d60: Concat/Stripe
    Size: 183750 blocks
    Stripe 0: (61250 blocks)
      Device            Start Block     Dbase
      c1t1d0s1               0           No
    Stripe 1: (61250 blocks)
      Device            Start Block     Dbase
      c1t2d0s1              1034         Yes
    Stripe 2: (61250 blocks)
      Device            Start Block     Dbase
      c1t3d0s1               0           No
      .
      .
      .
```

First, the metadevice output contains the device name of the metadevice, `/dev/md/dsk/d60`. The size of the metadevice in blocks is shown on the second line.

The components, which are shown as a concatenation of stripes, are shown on separate lines. Information about each component in the stripe is shown with a separate entry followed by the size, in blocks, of the stripe. Each entry consists of the “Device,” “Start Block,” and “Dbase.”

The “Device” is the name of the device that is in the stripe. The “Start Block” is the block on which the component begins. The “Dbase” field specifies whether or not a database resides on the component.

Note – State and hot spare information for stripes which are not submirrors is not retained by DiskSuite.

Mirroring Existing File Systems

Once the DiskSuite software package is installed, you can mirror existing file systems without backing up data or reconfiguring devices.

To mirror an existing file system, you must use an additional component of equal or greater size. It is possible to use a concatenation of two or more components that have adequate space available to contain the mirror. For example, if `/export/home` is mounted on a 1-Gbyte component named `/dev/dsk/c0t0d0s0`, at least 1-Gbyte of space must be available on the metadevice that is being defined for the new submirror.

When you mirror an existing file system, be sure that you initially configure a one-way metamirror with a submirror containing the existing file system. A submirror that you add subsequently with the `metattach` command should not contain any data that is needed, since it will overwritten by the resync following the `metattach` command. For a more detailed explanation of this process, see “Mirroring an Existing File System” in the “Examples” section at the end of this chapter.

Unmirroring File Systems

The procedure for unmirroring a file system that is not `root`, `swap`, or `/usr`, consists of the following steps:

1. Unmounting the file system
2. Running `metadetach(1M)` on the submirror that will continue to be used for the file system
3. Running `metaclear(1M)` on the mirror
4. Changing the file system entry in the `/etc/vfstab` file to use a nonmirror device, if the file system entry appears there
5. Removing the metamirror and submirror entries from the `/etc/opt/SUNWmd/md.tab` file
6. Remounting the file system

Example of Unmirroring a File System

The following is an example of unmirroring the `/var` file system. In this example, `/var` is made up of a two-way mirror with the names `d2` and `d3` in a mirror named, `d4`. The names of the components are `/dev/dsk/c0t0d0s0` and `/dev/dsk/c1t0d0s0`.

```
# /sbin/umount /var
# /usr/opt/SUNWmd/sbin/metadetach /dev/md/dsk/d4 /dev/md/dsk/d2
# /usr/opt/SUNWmd/sbin/metaclear -r /dev/md/dsk/d4
```

After you run the above commands, you must change the entry in the `/etc/vfstab` file (if one appears for `/var`). For example, the following line:

```
/dev/md/dsk/d4 /dev/md/rdsk/d4   /var   ufs   4   yes   –
```

should be changed to read:

```
/dev/md/dsk/d2 /dev/md/rdsk/d2   /var   ufs   4   yes   –
```

By using the metadevice name, `/dev/md/dsk/d2` instead of `/dev/md/dsk/d4`, you have performed the unmirroring. By using `/dev/md/dsk/d2` rather than the component name (for example, `/dev/dsk/c0t0d0s0`), you can continue to have the device support either a concatenation or a stripe.

The `/var` file system can be remounted.

```
# /sbin/mount /var
```

`root`, `swap`, *and* `/usr` *Mirroring*

The key feature that makes mirroring of the `root`, `swap`, and `/usr` file systems possible is that the state database locations are patched into the metadisk driver via the `/etc/system` file. Since the kernel has knowledge of the state database early in the boot process, it can obtain all the information it needs about all metadevices. This is explained further in Chapter 2, “Installation and Setup” and Chapter 8, “Disksets.” Additional information about the `mddb.cf` file is provided in Appendix A, “Solstice DiskSuite Files.”

Note – If problems occur when mirroring `root`, `swap`, and `/usr`, refer to Appendix C, "Recovery From Failed Boots."

Mirroring `/usr`

There are seven basic steps involved with mirroring the `/usr` file system. In the following example, `/dev/dsk/c0t3d0s6` is the existing device where `/usr` resides. The component used for the mirror of this file system is `/dev/dsk/c1t0d0s6`. The steps used to mirror `/usr` are:

1. **Specify a metadevice for the existing file system by making an entry in the `/etc/opt/SUNWmd/md.tab` file.**

```
/dev/md/dsk/d12 1 1 /dev/dsk/c0t3d0s6
```

2. **Specify the one-way mirror in the `/etc/opt/SUNWmd/md.tab` file by inserting the following line:**

```
/dev/md/dsk/d2 -m /dev/md/dsk/d12
```

3. **Specify a metadevice for the new submirror on the other half of the mirror by entering the following in the `/etc/opt/SUNWmd/md.tab` file:**

```
/dev/md/dsk/d22 1 1 /dev/dsk/c1t3d0s6
```

   The number of blocks in `/dev/dsk/c1t3d0s6` must be greater or equal to the number of blocks in `/dev/dsk/c0t3d0s6`. Otherwise, the `metattach` command in Step 7 of this procedure will fail.

4. **Run the `metainit` command for each of the metadevices and the metamirror defined in the `/etc/opt/SUNWmd/md.tab` file.**
   Note that a `-f` option is used with `metainit` for `/dev/md/dsk/d12` because the component being used has `/usr` currently mounted (see Step 1). `metainit` normally fails if the component has a file system currently mounted, but the `-f` force option allows the metadevice to be activated.

```
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d22
# /usr/opt/SUNWmd/sbin/metainit -f /dev/md/dsk/d12
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d2
```

5. **Edit the `/etc/vfstab` file to change the entry for the `/usr` file system to be the newly defined metadevice name rather than the component name.**
   For example, change the following line:

```
/dev/dsk/c0t3d0s6 /dev/rdsk/c0t3d0s6 /usr ufs 2 no -
```

to read:

```
/dev/md/dsk/d2 /dev/md/rdsk/d2 /usr ufs 2 no -
```

6. **Reboot the system using the `reboot(1M)` command.**

```
# /usr/sbin/reboot
```

7. **Attach the new metadevice to the one-way mirror, using the `metattach(1M)` command.**

```
# /usr/opt/SUNWmd/sbin/metattach /dev/md/dsk/d2 /dev/md/dsk/d22
```

Once the above steps are performed, a resync of the mirror begins automatically, which copies all the data that exists on `d12` (`/dev/dsk/c0t3d0s6`) to the submirror `d2` (`/dev/dsk/c1t3d0s6`).

Mirroring `root`

The procedure for mirroring root on an x86 system is somewhat different from the procedure used for a SPARC system. Use the procedure described in one of the following two sections to set up `root` mirroring on your system.

Caution – When you are mirroring the root file system it is essential that you record the secondary root slice name in order to reboot the system if the primary submirror fails. This information should be written down, not recorded on the system which may not be available. Read either the section, “Booting From the Alternate Device — SPARC System” on page 83 or “Booting From the Alternate Device — x86 System” on page 89 for details.

Mirroring `root` on a SPARC System

There are eight basic steps involved with mirroring the `root` file system on a SPARC system. In the following example, `/dev/dsk/c0t3d0s0` is the existing device where `root` resides. The component used to mirror this component is `/dev/dsk/c1t3d0s0`.

Setting Up a Root Mirror — SPARC System

Follow these steps to mirror `root` on a SPARC system:

1. **Create a metadevice for the existing file system by making an entry in the `/etc/opt/SUNWmd/md.tab` file.**

```
/dev/md/dsk/d10 1 1 /dev/dsk/c0t3d0s0
```

2. **Create the one-way mirror in the `/etc/opt/SUNWmd/md.tab` file by inserting the following line:**

```
/dev/md/dsk/d0 -m /dev/md/dsk/d10
```

3. **Create a metadevice for the new component on the other half of the mirror by entering the following in the `/etc/opt/SUNWmd/md.tab` file:**

```
/dev/md/dsk/d20 1 1 /dev/dsk/c1t3d0s0
```

4. **Run the** `metainit(1M)` **command for each of the metadevices and the metamirror defined in the** `/etc/opt/SUNWmd/md.tab` **file.**
   Note that a `-f` option is used with `metainit` for `d10` because the component being used has `root` currently mounted (see Step 1). `metainit` normally fails if the component has a file system currently mounted, but the `-f` force option allows the metadevice to be activated.

```
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d20
# /usr/opt/SUNWmd/sbin/metainit -f /dev/md/dsk/d10
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d0
```

5. **Run the** `metaroot(1M)` **command.** `metaroot` **edits the** `/etc/system` **and** `/etc/vfstab` **to add information necessary to mirror** `root`**.**

```
# /usr/opt/SUNWmd/sbin/metaroot /dev/md/dsk/d0
```

6. **Reboot the system using the** `reboot(1M)` **command.**

```
# /usr/sbin/reboot
```

7. **Attach the new metadevice to the one-way mirror, using the** `metattach` **command.**

```
# /usr/opt/SUNWmd/sbin/metattach /dev/md/dsk/d0 /dev/md/dsk/d20
```

Once the above steps are performed, a resync of the mirror begins automatically, which copies all the data that exists on `d10` (`/dev/dsk/c0t3d0s0`) to the submirror `d20` (the component named `/dev/dsk/c1t3d0s0`).

8. **Record the path to the alternate root device. You may need this later if the primary device fails.**
   In this example, you would determine the path to the alternate root device by entering:

```
# ls -l /dev/rdsk/c1t3d0s0
lrwxrwxrwx 1  root root  55 Mar 5 12:54  /dev/rdsk/c1t3d0s0 ->
../.
./devices/sbus@1,f8000000/esp@1,200000/sd@3,0:a
```

So you would record: `/sbus@1,f8000000/esp@1,200000/sd@3,0:a`

Booting From the Alternate Device — SPARC System

To boot your SPARC system from the alternate boot device, you would type:

```
# boot /sbus@1,f8000000/esp@1,200000/sd@3,0:a
```

Note – Backup Copilot users who are using system with open boot prom can use the OpenBoot `nvalias` command to define a “backup root” devalias for the secondary root mirror. For example:

ok **nvalias backup_root /sbus@1,f8000000/esp@1,200000/sd@3,0:a**

In the event of primary root disk failure, you then would only enter:

ok **boot backup_root**

Mirroring `root` *on an x86 System*

Before mirroring the `root` file system on an x86 system, you must first set up a Solaris partition large enough for your `root` mirror. For the purpose of the following procedures, we’ll assume that the alternate disk is `c0t1d0`.

Creating a Solaris Partition

Follow these steps to create a Solaris partition on an x86 system:

1. **Create the disk partition using the `fdisk` command as follows:**

```
# fdisk /dev/rdsk/c0t1d0p0
```

- If this is the first time you have run `fdisk`, the following is displayed:

```
The recommended default partitioning for your disk is:

a 100% “SOLARIS System” partition

To select this, please type “y”. To partition your disk
differently, type “n” and the “fdisk” program will let you
select other partitions.
```

- If you have previously run `fdisk`, you see a menu similar to the following:

```
        Total disk size is 1855 cylinders
        Cylinder size is 1110 (512 byte) blocks

Partition    Status    Type        Start    End    Length    %
=========    ======    ========    =====    ===    ======   ===
    1        Active    SOLARIS         1   1854     1854    100


SELECT ONE OF THE FOLLOWING:

     1.  Create a partition
     2.  Change Active (Boot from) partion
     3.  Delete a partition
     4.  Exit (Update disk configuration and exit)
     5.  Cancel (Exit without updating disk configuration)
Enter Selection:
```

2. **Select menu items to ensure that you have a Solaris partition large enough for your `root` mirror. Your Solaris partition should be five cylinders larger than the size needed to hold your `root` slice.**

Note – Make sure that the Solaris partition is active. Otherwise, you will be unable to boot from it.

---

3. **Use the** `format` **program to format the Solaris partition and create a slice for your** `root` **mirror as follows:**

```
# format
Searching for disks...done

AVAILABLE DISK SELECTIONS:
       0. c0t0d0 <drive type unknown>
          /eisa/eha@1000,0/cmdk@0,0
       1. c0t1d0 <DEFAULT cyl 1852 alt 2 hd 15 sec 74>
          /eisa/eha@1000,0/cmdk@1,0
       2. c0t2d0 <SUN0207 cyl 1254 alt 2 hd 9 sec 36>
          /eisa/eha@1000,0/cmdk@2,0
Specifiy disk (enter its number): 1

selecting c0t1d0
[disk formatted]


FORMAT MENU:
        disk       - select a disk
        type       - select (define) a disk type
        partition  - select (define) a partition table
        current    - describe the current disk
        format     - format and analyze the disk
        fdisk      - run the fdisk program
        repair     - repair a defective sector
        label      - write label to the disk
        analyze    - surface analysis
        defect     - defect list management
        backup     - search for backup labels
        verify     - read and display labels
        save       - save new disk/partition definitions
        inquiry    - show vendor, product and revision
        volname    - set 8-character volume name
        quit
```

4. Use the `partition` command to define a partition as follows:

```
format> partition

PARTITION MENU:
        0      - change ‘0' partition
        1      - change ‘1' partition
        2      - change ‘2' partition
        3      - change ‘3' partition
        4      - change ‘4' partition
        5      - change ‘5' partition
        6      - change ‘6' partition
        7      - change ‘7' partition
        select - select a predefined table
        modify - modify a predefined partition table
        name   - name the current table
        print  - display the current table
        label  - write partition map and label to the disk
        quit
partition> 0
Part       Tag     Flag    Cylinders         Size       Blocks
  0    unassigned   wm     0                 0          (0/0/0)

Enter partition id tag [unassigned]: root
Enter partition permission flags [wm]: wm
Enter new starting cyl [0]: 4
Enter partition size [0b, 0c, 0.00mb]: 400mb
partition> label
Ready to label disk, continue? y
partition>
```

5. **Exit from the** `partition` **menu and the** `format` **program by typing the** `quit` **command twice.**
   For example:

```
partition> quit

FORMAT MENU:
        disk       - select a disk
        type       - select (define) a disk type
        partition  - select (define) a partition table
        current    - describe the current disk
        format     - format and analyze the disk
        repair     - repair a defective sector
        label      - write label to the disk
        analyze    - surface analysis
        defect     - defect list management
        backup     - search for backup labels
        verify     - read and display labels
        save       - save new disk/partition definitions
        inquiry    - show vendor, product and revision
        volname    - set 8-character volume name
        quit
format> quit
```

Note the following important points about your Solaris `root` partition:

- Its id tag must be "`root`"
- Its size must be greater than or equal to the size of the original `root` partition
- It should not use cylinders 0-2

6. **Install the boot information on the alternate boot disk as follows:**

```
# installboot /usr/lib/fs/ufs/pboot \
  /usr/lib/fs/ufs/bootblk \
  /dev/rdsk/c0t1d0s2
```

Setting Up a Root Mirror — x86 System

After you have completed the above procedure for creating a Solaris partition, you can mirror the `root` file system. The steps to mirror `root` on an x86 system are as follows:

1. **Create a metadevice for the existing file system by making an entry in the `/etc/opt/SUNWmd/md.tab` file.**

```
/dev/md/dsk/d10 1 1 /dev/dsk/c0t0d0s0
```

2. **Create the one-way mirror in the `/etc/opt/SUNWmd/md.tab` file by inserting the following line:**

```
/dev/md/dsk/d0 -m /dev/md/dsk/d10
```

3. **Create a metadevice for the new component on the other half of the mirror by entering the following in the `/etc/opt/SUNWmd/md.tab` file:**

```
/dev/md/dsk/d20 1 1 /dev/dsk/c1t0d0s0
```

4. **Run the `metainit(1M)` command for each of the metadevices and the metamirror defined in the `/etc/opt/SUNWmd/md.tab` file.**
   Note that a `-f` option is used with `metainit` for `d10` because the component being used has `root` currently mounted (see Step 1). `metainit` normally fails if the component has a file system currently mounted, but the `-f` force option allows the metadevice to be activated.

```
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d20
# /usr/opt/SUNWmd/sbin/metainit -f /dev/md/dsk/d10
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d0
```

5. **Run `metaroot(1M)` command. `metaroot` edits the `/etc/system` and `/etc/vfstab` to add information necessary to mirror `root`.**

```
# /usr/opt/SUNWmd/sbin/metaroot /dev/md/dsk/d0
```

6. **Reboot the system using the `reboot(1M)` command.**

```
# /usr/sbin/reboot
```

7. **Attach the new metadevice to the one-way mirror, using the `metattach` command.**

```
# /usr/opt/SUNWmd/sbin/metattach /dev/md/dsk/d0 /dev/md/dsk/d20
```

   Once the above steps are performed, a resync of the mirror begins automatically, which copies all the data that exists on `d10` (`/dev/dsk/c0t0d0s0`) to the submirror `d20` (the component named `/dev/dsk/c1t0d0s0`).

8. **Record the path to the alternate `root` device. You may need this later if the primary device fails.**
   In this example, you would determine the path to the alternate `root` device by:

```
# ls -l /dev/rdsk/c1t0d0s0
lrwxrwxrwx 1  root root  55 Mar 5 12:54  /dev/rdsk/c1t0d0s0 ->
../.
./devices/eisa/eha@1000,0/cmdk@1,0:a
```

   So you would record: `/eisa/eha@1000,0/cmdk@1,0:a`

Booting From the Alternate Device — x86 System

To boot your x86 system from the alternate boot device, complete the following steps:

1. **Boot your system from the Multiple Device Boot (MDB) diskette.**
   After a moment, a screen similar to the following is displayed:

```
Solaris/x86 Multiple Device Boot Menu
Code    Device     Vendor     Model/Desc             Rev
============================================================

10      DISK       COMPAQ     C2244                  0BC4
11      DISK       SEAGATE    ST11200N SUN1.05       8808
12      DISK       MAXTOR     LXT-213S SUN0207       4.24
13      CD         SONY       CD-ROM CDU-8812        3.0a
14      NET        SMC/WD     I/O=300 IRQ=5
80      DISK       First IDE drive (Drive C:)
81      DISK       Second IDE drive (Drive D:)

Enter the boot device code:11
```

2. **Select your alternate disk from the above screen.**
   The following is displayed:

```
Solaris 2.4 for x86             Secondary Boot Subsystem,vsn 2.11

                  <<<Current Boot Parameters>>>
Boot path:/eisa/eha@1000,0/cmdk@0,0:a
Boot args:/kernel/unix

Type b[file-name] [boot-flags] <ENTER>     to boot with options
or   i<ENTER>                           to enter boot interpreter
or   <ENTER>                               to boot with defaults

                    <<<timeout in 5 seconds>>>
```

3. **Type `i` to select the interpreter.**

4. **Enter the following commands:**

```
>setprop boot-path /eisa/eha@1000,0/cmdk@1,0:a
>^D
```

Mirroring swap

There are seven basic steps involved with mirroring the swap partition. In the following example, /dev/dsk/c0t0d0s1 is the existing device where swap resides. The component used to mirror this component is /dev/dsk/c1t0d0s1. The steps used to mirror swap are:

1. **Create a metadevice for the existing file system by making an entry in the /etc/opt/SUNWmd/md.tab file.**

```
/dev/md/dsk/d11 1 1 /dev/dsk/c0t0d0s1
```

2. **Create the one-way mirror in the /etc/opt/SUNWmd/md.tab file by inserting the following line:**

```
/dev/md/dsk/d1 -m /dev/md/dsk/d11
```

3. **Create a metadevice for the new component on the other half of the mirror by entering the following in the /etc/opt/SUNWmd/md.tab file:**

```
/dev/md/dsk/d21 1 1 /dev/dsk/c1t0d0s1
```

4. **Run the metainit command for each of the metadevices and the metamirror defined in the /etc/opt/SUNWmd/md.tab file.**
   Note that the -f option is used with metainit for d10 because the component being used is currently being used as a swap device.

```
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d21
# /usr/opt/SUNWmd/sbin/metainit -f /dev/md/dsk/d11
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d1
```

5. **If an entry exists in the `/etc/vfstab` file that matches the partition on which the system was swapping, the file must be edited. (If such an entry does not exist, do not add one.) Change the name of the partition to the newly defined metadevice name rather than the partition name.**
   For example, change the following line:

```
/dev/dsk/c0t3d0s1 - - swap - no -
```

   to read:

```
/dev/md/dsk/d1 - - swap - no -
```

6. **Reboot the system using the `reboot(1M)` command.**

```
# /usr/sbin/reboot
```

7. **Attach the new metadevice to the one-way mirror, using the `metattach` command.**

```
# /usr/opt/SUNWmd/sbin/metattach /dev/md/dsk/d1 /dev/md/dsk/d21
```

Once these steps are performed, a resync of the mirror begins automatically. The resync copies all the data that exists on `d11` (`/dev/dsk/c0t0d0s1`) to the submirror `d21` (the component named `/dev/dsk/c1t0d0s1`).

Unmirroring `root` *and* `swap` *File Systems*

To unmirror any file system that is not `root` or `swap`, follow the instructions provided in “Unmirroring File Systems” on page 77. The procedure for unmirroring the `root` or `swap` file systems consists of six basic steps:

1. Making the metamirror that contains `root`, `swap`, or `/usr` into a one-way mirror
2. Changing the file system entry in the `/etc/vfstab` file to use a non-DiskSuite device
3. If you’re unmirroring `root`, running the `metaroot` command
4. Rebooting the system

5. Using the `metaclear` command, to clear the metamirrors and submirrors that were used by `root`, `swap`, or `/usr`
6. Removing the entries from the `/etc/opt/SUNWmd/md.tab` file

Example of Unmirroring Both `root` *and* `swap`

The following is an example of unmirroring both the `root` and `swap` file systems. In this example, `root` is a two-way mirror (`/dev/md/dsk/d0`) with submirrors `d10` and `d20`, which are made up of the components named `/dev/dsk/c0t3d0s0` and `/dev/dsk/c1t3d0s0`, respectively. `swap` is a two-way mirror (`/dev/md/dsk/d1`) with submirrors `d11` and `d21`, which are made up of the components named `/dev/dsk/c0t3d0s1` and `/dev/dsk/c1t3d0s1`, respectively.

1. **Make the metamirrors into a one-way mirror by using the `metadetach(1M)` command.**

```
# /usr/opt/SUNWmd/sbin/metadetach /dev/md/dsk/d0 /dev/md/dsk/d20
# /usr/opt/SUNWmd/sbin/metadetach /dev/md/dsk/d1 /dev/md/dsk/d21
```

   The submirrors are detached at this time because the machine is currently using the mirrors, thus the mirrors can not be cleared (using `metaclear`) at this time. By detaching and making the mirror one-way, no resync can occur, which could cause incorrect data to be written to the component that will be used following the unmirroring.

2. **Run the `metaroot(1M)` command which edits the `/etc/system` and `/etc/vfstab` files to remove information specifying the mirroring `root`.**

```
# /usr/opt/SUNWmd/sbin/metaroot /dev/dsk/c0t3d0s0
```

3. **Edit the** `/etc/vfstab` **file to change the entry for the** `swap` **file system.**
   The entry must be changed to remove the metadevice name and replace it with the component name. For example, change the following line:

```
/dev/md/dsk/d1 - - swap - no -
```

   to read:

```
/dev/dsk/c0t3d0s1 - - swap - no -
```

4. **Reboot the system using the** `reboot(1M)` **command.**

```
# /usr/sbin/reboot
```

5. **Use the** `metaclear` **command to clear the metadevices.**

```
# /usr/opt/SUNWmd/sbin/metaclear -r /dev/md/dsk/d0
# /usr/opt/SUNWmd/sbin/metaclear -r /dev/md/dsk/d1
# /usr/opt/SUNWmd/sbin/metaclear /dev/md/dsk/d20
# /usr/opt/SUNWmd/sbin/metaclear /dev/md/dsk/d21
```

6. **Remove all entries for** `root` **and** `swap` **from the** `/etc/opt/SUNWmd/md.tab` **file.**

Reconfiguring Submirrors

Several DiskSuite commands are available that make the reconfiguration of submirrors possible with little or no disruption of service. The functionality includes utilities to attach or detachsubmirrors, or bring submirrors online and offline. The replacement of components within submirrors is also explained. The uses of these utilities are explained in the following sections. For a complete description of all the options associated with the commands, refer to Appendix E, “Using Solstice DiskSuite 4.0 with the SPARCstorage Array 100.”

Attaching and Detaching Submirrors

Metadevices (concatenations or stripes) are attached to a metamirror as a submirror beneath the metamirror with the `metattach(1M)` command. The DiskSuite software supports up to three-way mirroring, thus a submirror can be attached only if fewer than three are already attached to the metamirror.

After a new metadevice is attached, `metattach` automatically starts a resync operation to the newly added submirror. `metattach` can also be used to add components to an existing concatenated metadevice without interrupting service.

Metadevices are separated (detached) from the metamirror with the `metadetach(1M)` command. Once detached, the metadevice is no longer a part of the metamirror. Reads from and writes to the metamirror no longer go to the detached metadevices. However, the metadevice can be used for other purposes. To help protect your data, DiskSuite does not allow the last remaining submirror to be detached.

An example of detaching a metadevice might occur when errors are being reported. You could run `metadetach` to detach that submirror component without disruption of service from the system. The downtime to replace the drive could be scheduled at a more convenient time.

Placing Submirrors Online and Offline

The DiskSuite commands, `metaonline(1M)` and `metaoffline(1M)` are used to place submirrors online and offline.

These commands are useful when, for instance, one component in a physical SCSI chain fails. In this case, all other components on the chain could be taken offline while the broken component is replaced. After replacing the component, the other components in the SCSI chain can be brought back online, using `metaonline`.

When `metaoffline` is used on a submirror, the metadisk is prevented from reading from and writing to the submirror. While the submirror is offline, DiskSuite keeps track of all writes to the metamirror and they are written to the submirror when it is brought back online.

The `metaoffline` command's functionality is similar to that offered by `metadetach`, however `metaoffline` does not sever the logical association between the submirror and the metamirror.

`metaonline` can be used only when a submirror was taken offline using the `metaoffline` command. When `metaonline` is run, reading from and writing to the submirror resumes. A resync is automatically performed to resync the regions of the metamirror that were written to while the submirror was offline. Writes are directed to the submirror during the resync. Reads, however, will come from a different submirror during the resync operation. Once the resync is complete, reads and writes are performed on the submirror previously taken offline.

Replacing and Enabling Submirror Components

Before you replace or enable a component, you should first check to be sure it is partitioned correctly. See the `fmthard(1M)` and `prtvtoc(1M)` man pages for details.

Note – The partitioning information for any disk used by DiskSuite should be saved in a safe place before any errors occur.

A component being used as a replacement in a submirror must be as large as the component it is replacing.

If a component is in the "Last Erred" state, you cannot replace it until you first replace all of the other mirrored components in the "Maintenance" state. Remember, however, that replacing or enabling a component in the "Last Erred" state usually means that some data has been lost. Be sure to validate the data on the mirror be reusing it (see `fsck(1M)`). If components are in the "Maintenance" state, no data has been lost and you can safely replace or enable components in any order.

Always check the metadevice state database replicas (using `metadb -i`) when repairing components. Any database replica shown to be in error should be deleted (using `metadb -d`) and added back (using `metadb -a`) to repair it.

You can replace components within a submirror with the `metareplace(1M)` command. `metareplace` automatically starts a resync to get the new component in sync with the rest of the metamirror.

Enabling components within a submirror is also performed using the `metareplace` command. `metareplace -e` changes the state of a failed component and automatically starts a resync to get the component in sync with the rest of the metamirror.

This command is useful when a component fails due to human error (for example, write-protecting an active disk), because a cable was loose and retightened, or because the component was physically replaced. When a component is replaced, the component used for replacement must have the same attributes as the component that was replaced. For example, if SCSI target 3 (`/dev/dsk/c2t3d0s1`) was physically replaced with a disk that was also SCSI target 3, then `metareplace -e` should be used to enable the component.

Changing Metamirror and Submirror Options

You can use the `metaparam(1M)` command to change most submirror options while DiskSuite is running. The options that can be changed include the read and write options, pass numbers, and hot spare pools.

`metaparam` allows you to change all metamirror options from the command line, with the exception of the interlace value. To change the interlace value, you must run the `metainit` command. On existing metadevices the `metaclear` command must be run prior to using `metainit` to change the interlace value.

The options are the same as those reported by either the `metastat` or the `metaparam` commands. For example:

```
# /usr/opt/SUNWmd/sbin/metaparam /dev/md/dsk/d25
d25: metamirror
Pass = 1
Read option = round robin
Write option = parallel
```

Entering the following command would change the read option to geometric, the write option to serial, and the pass number to five.

```
# /usr/opt/SUNWmd/sbin/metaparam -r geometric \
          -w serial -p 5 /dev/md/dsk/d25
```

Note – The above command will not change any entry in the `/etc/opt/SUNWmd/md.tab` file. However the information is automatically modified in the state database and also in the `/etc/opt/SUNWmd/md.cf` file.

Using Mirrors for Online Backup

Mirroring enables you to perform online backups. Because each submirror is an exact copy of the file system, it can be taken offline and backed up to tape. This eliminates taking the system down to make a backup of the system. With the DiskSuite software package, file system locking functionality also eliminates the need to run `fsck(1M)` after taking the metadevice offline.

There are drawbacks, however, to using mirrors for backup purposes. For instance, if the backup is of a two-way mirror, all data redundancy is lost while one submirror is taken offline for backup. This problem can be overcome by adding hardware to create a mirror containing three submirrors. There is also the overhead of bringing the submirror back online after the backup is complete.

If your site uses procedures for online backup like the ones described in the following sections, you should combine the commands into a script.

The following sections provide instructions for performing online backups of both mirrored and non-mirrored metadevices.

Performing Online Backups — for Mirrors

A mirrored metadevice can be backed up (archived) without unmounting it or taking the entire mirror offline. One of the submirrors must be taken offline temporarily, thus losing mirroring, but it can be placed online and resynced as soon as the backup is complete — without halting the system or denying user access to the data.

There are five steps in the procedure for backing up a file system that is being mirrored. The steps listed here are described in detail in the example below.

1. Write locking the file system
2. Using the `metaoffline` command to take one metadevice offline from the metamirror

3. Unlocking the file system
4. Backing up the data on the offlined metadevice
5. Using the `metaonline` command to place the offlined metadevice back online

Using mirrors to perform online backups creates a backup that is a "snapshot" of an active file system. For example, a problem will occur if a user is editing a file with `vi` and uses a `:w` immediately before the `lockfs` command is run. In this case, since the `vi` file is being written from the top, it is possible that the file will appear completely empty on the backup. It is also possible that only the first 100 characters of a 10 Kbyte text file are written to the backup.

In the example procedure that follows, the metamirror (`/dev/md/dsk/d1`) consists of the three metadevices (`/dev/md/dsk/d2`, `/dev/md/dsk/d3`, and `/dev/md/dsk/d4`).

Note – Do not perform this procedure on the root file system.

To take one side of the mirror offline and perform a backup, you would follow these steps:

1. **Use `lockfs` with the `-w` option to lock the file system from writes.**
   You need to lock the file system only if a UFS file system resides on the metamirror. For example, if the metadevice is set up as a raw device for database management software or some other specific application, it would not be necessary to use `lockfs`. (You may, however, want to run the appropriate vender-dependent utility to flush any buffers and lock access.)

```
# /usr/sbin/lockfs -w mount_point
```

2. **Take one submirror offline from the metamirror.**

```
# /usr/opt/SUNWmd/sbin/metaoffline /dev/md/dsk/d1 /dev/md/dsk/d4
```

Reads will continue to be made from `/dev/md/dsk/d2` and `/dev/md/dsk/d3`. However, `/dev/md/dsk/d4` will be out of sync as soon as the first write is made to the metamirror. This inconsistency is corrected when `/dev/md/dsk/d4` is onlined in Step 5.

You don't need to run `fsck` on the offlined file system. The only possible inconsistency would be that files that have been deleted but were open at the time `lockfs -w` was run would still occupy file space on the file system.

3. **Use `lockfs` with the `-u` option to unlock the file systems and allow writes to continue.**
(You may need to perform necessary unlocking procedures based on vendor-dependent utilities used in Step 1 above.)

```
# /usr/sbin/lockfs -u mount_point
```

4. **Perform a backup, using `ufsdump` or whatever other backup utility is normally used to copy the offlined metadevice to tape or another medium.**

Note – To ensure a proper backup, make certain you use the *raw* device (in this case, `/dev/md/rdsk/d4`). Using "`rdsk`" allows greater than 2GB access.

5. **Place the metadevice back online using the `metaonline` command.**

```
# /usr/opt/SUNWmd/sbin/metaonline /dev/md/dsk/d1 /dev/md/dsk/d4
```

When `/dev/md/dsk/d4` is placed online, it is automatically resynced with the metamirror.

Performing Online Backups — for Nonmirrors

DiskSuite allows you to attach or detach submirror while the product is operating. This functionality can be used to temporarily attach another metadevice to the one-way mirror, take the mirror offline, and then perform the backup without interrupting service to users.

While this approach to online backup is not the usual use for mirroring (since it is not being used to provide replication of data), it does offer a method of performing online backups.

In the following example, a server has four disk drives attached. Three of the components are configured as one-way mirrors, while the fourth component is defined as an unused (spare) metadevice. The spare metadevice is attached to each of the mirrors in turn and a backup is performed.

Note – This is not a recommended configuration or use of DiskSuite. This example is included only to illustrate a method of performing online backups.

The steps to use the mirroring facility to back up the file system are:

1. Making sure all the metadevices and metamirrors are defined
2. Defining the spare metadevice
3. Attaching the spare metadevice to the metamirror and waiting for the resync to finish
4. Using `lockfs -w` to write-lock the file system.
5. Taking the spare metadevice offline
6. Using `lockfs -u` to unlock the file system
7. Performing the backup
8. Placing the spare metadevice back online
9. Clearing the spare metadevice

In the example procedure that follows, `/dev/md/dsk/d11` is the spare metadevice, and `/dev/md/dsk/d1` is the metamirror to be backed up.

Complete the following procedure to backup a file system using the mirroring facility:

1. **Activate the spare metadevice.**
   This is not necessary if the spare metadevice already active.

```
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d11
```

2. **Attach the spare metadevice to a metamirror.**
   When you run the `metattach` command, a resync is automatically started. (This may take some time to complete.)

```
# /usr/opt/SUNWmd/sbin/metattach /dev/md/dsk/d1 /dev/md/dsk/d11
```

3. **Wait for the device to synchronize.**
   Using the `metastat` command periodically as follows, you'll be able to tell when the synchronization is complete.

```
# metastat d1
```

4. **Use the `lockfs` command with the `-w` option to lock the file system from writes.**

```
# /usr/sbin/lockfs -w mount_point
```

   You need to lock the file system only if a UFS file system resides on the metamirror. For example, if the metadevice is set up as a raw device for database management software or some other specific application, it would not be necessary to use `lockfs` (though you may want to synchronize data in some other vender-dependent way). In this case, however, it may be desirable to take the database offline temporarily to synchronize the data.

5. **Take the spare metadevice offline from the metamirror using the `metaoffline` command.**

```
# /usr/opt/SUNWmd/sbin/metaoffline /dev/md/dsk/d1 /dev/md/dsk/d11
```

6. **Use `lockfs` with the `-u` option to unlock the file systems and allow writes to continue.**

```
# /usr/sbin/lockfs -u mount_point
```

7. **Perform the backup, using `ufsdump` or whatever other backup utility is normally used to copy the detached metadevice to tape or another medium.**

Note – To ensure a proper backup, make certain you use the *raw* device (in this case, `/dev/md/rdsk/d11`). Using "`rdsk`" allows greater than 2GB access.

8. Place the spare metadevice back online.

```
# /usr/opt/SUNWmd/sbin/metaonline /dev/md/dsk/d1 /dev/md/dsk/d11
```

9. Detach the spare metadevice.

```
# /usr/opt/SUNWmd/sbin/metadetach /dev/md/dsk/d1 /dev/md/dsk/d11
```

10. Clear the spare metadevice.

```
# /usr/opt/SUNWmd/sbin/metaclear /dev/md/dsk/d11
```

Examples

Several examples are offered in this section that show, in a step-by-step fashion, how to set up mirrors in various configurations. These examples include:

- Mirroring an existing file system
- Adding submirrors
- Watching the progress of a resync using `metastat`

Mirroring an Existing File System

This example shows how to set up a mirror of the file system located on `/dev/dsk/c1t0d0s0`. This example is typical for any existing file systems, except `/usr`, `root`, and `swap`. The steps to follow are:

1. **Create two metadevices for the existing file system by making the following entries in the `/etc/opt/SUNWmd/md.tab` file.**
   The first two entries create the metadevices, named `/dev/md/dsk/d1` and `/dev/md/dsk/d2`, that will be used as submirrors. The third line creates a one-way metamirror, using only the first metadevice. The first metadevice is the existing component which contains the `/master` file system.

```
/dev/md/dsk/d1 1 1 /dev/dsk/c1t0d0s0
/dev/md/dsk/d2 1 1 /dev/dsk/c2t0d0s0
/dev/md/dsk/d0 -m /dev/md/dsk/d1
```

2. **Unmount the file system.**

```
# /sbin/umount /master
```

   If the file system is busy, an error message is displayed.

3. **Initialize the metadevices and metamirror.**

```
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d1
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d2
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d0
```

4. **Edit the `/etc/vfstab` file to change the entry for `/master` to be `/dev/md/dsk/d0`.**
   For example, if the entry was:

```
/dev/dsk/c1t0d0s0 /dev/rdsk/c1t0d0s0 /master ufs 5 yes -
```

   change the entry to read:

```
/dev/md/dsk/d0 /dev/md/rdsk/d0 /master ufs 5 yes -
```

5. **Mount the `/master` file system.**
   When it is mounted, it is a one-way mirror.

```
# /sbin/mount /master
```

6. **Now attach the second metadevice, `/dev/md/dsk/d2`, to the mirror `/dev/md/dsk/d0`, using the `metattach` command.**

```
# /usr/opt/SUNWmd/sbin/metattach /dev/md/dsk/d0 /dev/md/dsk/d2
```

When you run the `metattach` command, a resync of the mirror is started. Once the resync completes, `/master` is mirrored.

Adding Submirrors

In the following example, two submirrors are being added to the metamirror, `/dev/md/dsk/d30`. In this example, the metamirror already exists.

The metamirror, `/dev/md/dsk/d30`, appears as follows when you run `metastat`:

```
# /usr/etc/metastat /dev/md/dsk/d30
d30: metamirror
  Submirror 0: d15
  State: Okay
  Regions which are dirty: 0%
  Pass = 1
  Read option = geometric (-g)
  Write option = parallel (default)
  Size: 183750 blocks
  .
  .
  .
```

Note the size in the `metastat` output is 183750 blocks, thus each of the two submirrors being added must be 183750 blocks or larger to be added as submirrors. If the submirrors are larger, the extra space will be unused and unavailable.

Follow these steps to add the two submirrors from the command line:

1. **Use** `metainit` **to create the two new submirrors.**
   For example:

```
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d70 1 1  \
  /dev/dsk/c0t3d0s0
# /usr/opt/SUNWmd/sbin/metainit /dev/md/dsk/d80 1 1  \
  /dev/dsk/c1t2d0s0
```

2. **Use the** `metattach` **command to attach each of the new submirrors to the metamirror.**
   When `metattach` is run, the new submirrors are synced with the existing metamirror.

```
# /usr/opt/SUNWmd/sbin/metattach /dev/md/dsk/d30 /dev/md/dsk/d70
# /usr/opt/SUNWmd/sbin/metattach /dev/md/dsk/d30 /dev/md/dsk/d80
```

   If you wish to view the metamirror after adding the two submirrors, you can run the `metastat` command again. The output in this example would appear as follows:

```
# /usr/etc/metastat /dev/md/dsk/d30
d30: metamirror
  Submirror 0: d15
  State: Okay
  Submirror 1: d70
  State: Okay
  Submirror 2: d80
  State: Okay
  Regions which are dirty: 0%
  Pass = 1
  Read option = geometric (-g)
  Write option = parallel (default)
  Size: 183750 blocks
  .
  .
  .
```

Watching the Progress of a Resync Using `metastat`

In this example, the metadevice `/dev/md/dsk/d22` is being attached to the metamirror `/dev/md/dsk/d21`.

Follow these steps to attach the metadevice:

1. **Define the metadevice either in the** `/etc/opt/SUNWmd/md.tab` **file or on the command line.**
2. **Run** `metainit` **on the metadevices.**
3. **Use** `metattach` **to attach the new metadevices to the metamirror.**

```
# /usr/opt/SUNWmd/sbin/metattach /dev/md/dsk/d21 /dev/md/dsk/d22
```

4. **Use the** `metastat` **command with the metamirror name as an argument to view the status of the new metadevice.**

```
# /usr/etc/metastat /dev/md/dsk/d21
d21: metamirror
  Submirror 0: d30
  State: Okay
  Submirror 1: d22
  State: Resyncing
  Regions which are dirty: 3%
  Resync in progress: 30% done
  Pass = 1
  Read option = geometric (-g)
  Write option = parallel (default)
  Size: 183750 blocks
```

To view the progress, you can run the `metastat` command again.

```
# /usr/etc/metastat /dev/md/dsk/d21
d21: metamirror
  Submirror 0: d30
  State: Okay
  Submirror 1: d22
  State: Resyncing
  Regions which are dirty: 6%
  Resync in progress: 50% done
  .
  .
  .
```

5

UFS Logging 6

DiskSuite's UFS logging facility speeds up reboots, provides faster local directory operations, and decreases synchronous disk writes. UFS file system updates are safely recorded in a log before they are applied to the file system itself.

This chapter provides information on how to use DiskSuite's UFS logging facility. Use the following table to locate specific information.

6

Overview of UFS Logging

UFS file systems are checked at boot time because unscheduled system downtime can interrupt file system updates. These partially completed updates can leave inconsistences in a file system. Mounting a file system without first checking it and repairing any inconsistencies can cause panics or data corruption. Checking consistency for large file systems can be a time consuming process. With the UFS logging feature, file systems do not have to be checked at boot time because the changes from unfinished system calls are discarded.

A pseudo device, called the *metatrans device*, is responsible for managing the contents of the log. Like other metadevices, the metatrans device behaves the same as an ordinary disk partition. The metatrans device is made up of two subdevices: the *logging device* and the *master device*. These can be disk partitions, metadevices, or metamirrors, but not metatrans devices.

The logging device can be shared by several file systems. The logging device contains the log. This log is a sequence of records, each of which describes a change to a file system. The master device contains the file system itself. The master device can either contain a file system initially, or you can create a file system on the metatrans device. Logging begins automatically when the metatrans device is mounted.

Setting Up UFS Logging

You set up and configure a metatrans device using the standard DiskSuite utilities. The configuration of the device and other state information is stored in the metadevice state database. DiskSuite's dynamic concatenation facility provides dynamic concatenation of both the master and logging devices.

Metatrans devices can be defined either from the command line or in the `/etc/opt/SUNWmd/md.tab` file, and are given names such as `/dev/md/dsk/d1`. This is the same naming convention used for other metadevices.

After a metatrans device is configured, it can be used just as if it were a physical partition. This means it can be used as a device (up to one terabyte). A file system can be created on the metatrans device if the master device doesn't already have a file system. Most UNIX disk utilities will work normally on metatrans devices, with the exception of `format`(1M).

In addition, all usual file system operations can be performed on a metatrans device. The following list offers examples of file system operations:

- `mount`(1M) the metatrans device on a directory
- `umount`(1M) a mounted metatrans device
- Copy files to the metatrans device
- Read and write files from and to the metatrans device
- `ufsdump`(1M) and `ufsrestore`(1M) the metatrans device

The following sections provide answers to some of the questions you'll need to resolve before you begin setting up UFS logging on your system.

How Much Log Space is Required?

The minimum size for a logging device is 1 Mbyte of disk space. Larger logs allow for greater concurrency (more simultaneous file system operations per second). As a general rule, you need about 1Mbyte of log space for every 100 Mbyte of file system space being logged.

Note – Logs larger than 64 Mbyte are a waste of disk space.

Which File Systems Should Be Logged?

It is possible to log any UFS file system with the exception of `root` (`/`). The `root` file system cannot be a metatrans device. In addition, it is probably not necessary to log small file systems with mostly read activity.

Warning – When you are logging `/usr`, `/var`, `/opt`, or any other system file systems used during a Solaris upgrade/installation, logging must be disabled before you can perform a Solaris upgrade/installation.

In general, you will want to log your largest UFS file systems and the UFS file systems whose data is changed most frequently.

All logged file systems can share the same log. For performance considerations, however, the file systems with the heaviest loads should have separate logs.

Note – It is strongly recommended that you mirror all logs. Losing the data in a log to device errors can leave a file system in a corrupt state.

Where Should the Log Be Located?

Before you begin the procedures for setting up UFS logging, you should determine where you want to place the logs. You have two options for placement of logs:

- On unused partitions
- On the partitions containing the state databases

The second of these options can be a practical solution if you are unable to commit an unused partition to logging. See Chapter 8, “Disksets,” for further information on using the state database with DiskSuite and Chapter 2, “Installation and Setup” for information on creating space for metadevice state database replicas.

How To Set Up UFS Logging

After you have determined which file systems you want to log and where the log will be located, follow the steps in this section to set up UFS logging.

There are two methods of defining a metatrans device:

- Edit the `md.tab` file to define the metatrans device, then run the `metainit(1M)`command; or
- Include the definition of the metatrans device on the command line with the `metainit(1M)`command.

For a complete discussion of the `md.tab` file, refer to Appendix A, “Solstice DiskSuite Files.”

In the following example, the metatrans device is defined from the command line. For the purpose of this example, we’ll assume that the partition you’ve selected for logging is `/dev/dsk/c2t2d0s1` and you want to enable logging on `/abcfs`. Assume the `/etc/vfstab` entry for `/abcfs` is:

```
/dev/dsk/c0t2d0s6   /dev/rdsk/c0t2d0s6   /abcfs   ufs   5   yes   –
```

In this case, you would do the following to set up UFS logging on /abcfs:

1. **As** root**, use the** umount **command to unmount** /abcfs:

```
# umount /abcfs
```

2. **Use the** metainit **command to create the metatrans device:**

```
# metainit d64 -t c0t2d0s6 c2t2d0s1
```

This creates the metatrans device /dev/md/dsk/d64. This metatrans device is composed of the master device c0t2d0s6 and the logging device c2t2d0s1.

3. **Edit the** /etc/vfstab **file to include the new metatrans device.**
In this case, you would change the line:

```
/dev/dsk/c0t2d0s6   /dev/rdsk/c0t2d0s6   /abcfs   ufs   5   yes   –
```

to read:

```
/dev/md/dsk/d64   /dev/md/rdsk/d64   /abcfs   ufs   5   yes   –
```

4. **Use the** mount **command to mount** /abcfs:

```
# mount /abcfs
```

/abcfs is now a logging file system. When you reboot the system, fsck will notice that this is a logging file system and will not check it. Instead, fsck will display:

```
/dev/md/rdsk/d64: is logging.
```

How to Share a Log Between File Systems

It may be practical in your situation to share a log among several or all file systems. This section provides the procedure for sharing a log between file systems.

In this example, we'll assume that you have set up `/abcfs` as described in the previous section and you want to set up `/xyzfs` to share the same log as `/abcfs`. Assume that the entry in the `/etc/vfstab` file for `/xyzfs` is:

```
/dev/dsk/c0t2d0s4   /dev/rdsk/c0t2d0s4   /xyzfs  ufs   5   yes   –
```

In this case, you would do the following to set up `/xyzfs` to share the same log with `/abcfs`:

1. **Use the `umount` command to unmount `/xyzfs`.**

```
# umount /xyzfs
```

2. **Use the `metainit` command to create the metatrans device:**

```
# metainit d63 -t c0t2d0s4 c2t2d0s1
```

   This creates the metatrans device `/dev/md/dsk/d63`. This metatrans device is composed of the master device `c0t2d0s4` and the now shared logging device `c2t2d0s1`.

3. **Edit the `/etc/vfstab` file to include the new metatrans device.**
   In this case, you would change the line:

```
/dev/dsk/c0t2d0s4   /dev/rdsk/c0t2d0s4   /xyzfs  ufs   5   yes   –
```

   to read:

```
/dev/md/dsk/d63   /dev/md/rdsk/d63   /xyzfs   ufs   5   yes   –
```

4. **Use the** `mount` **command to mount** `/xyzfs`**:**

```
# mount /xyzfs
```

5. `/xyzfs` **is now a logging file system. When you reboot the system,** `fsck` **will notice that** `/abcfs` **and** `/xyzfs` **are logging file systems and will not check them. Instead,** `fsck` **will display:**

```
/dev/md/rdsk/d64: is logging.
/dev/md/rdsk/d63: is logging.
```

Removing UFS Logging

If you decide you want to remove UFS logging from a particular file system, use the steps in this section to complete the procedure.

Assume that the current `/etc/vfstab` entry for `/abcfs` is as follows:

```
/dev/md/dsk/d64   /dev/md/rdsk/d64   /abcfs   ufs   5   yes   –
```

To remove UFS logging from /abcfs, you would do the following:

1. **Use the** `umount` **command to unmount** `/abcfs`**:**

```
# umount /abcfs
```

2. **Use the** `metaclear` **command to clear the metatrans device:**

```
# metaclear d64
```

This clears the metatrans device `/dev/md/dsk/d64`. Any information pertaining to the master device is rolled from the log prior to clearing the device.

3. **Edit the** `/etc/vfstab` **file to remove the metatrans device information.**
   In this case, you would change the line:

```
/dev/md/dsk/d64   /dev/md/rdsk/d64   /abcfs   ufs   5   yes   -
```

   to read:

```
/dev/dsk/c0t2d0s6   /dev/rdsk/c0t2d0s6   /abcfs   ufs   5   yes   -
```

4. **Because** `/abcfs` **is no longer a logging file system, you must run** `fsck` **before you can mount it. The following output is generated when you run** `fsck`**:**

```
# fsck /dev/rdsk/c0t2d0s6
FILE SYSTEM STATE IN SUPERBLOCK IS WRONG; FIX? y
```

   The correct response for this output is “y”.

5. **Mount** `/abcfs` **using the** `mount` **command:**

```
# mount /abcfs
```

   The `/abcfs` file system is no longer logging.

Logging a File System That Cannot Be Unmounted

You may want to log a file system that cannot be unmounted. An example of this is `/usr`.

Using the `/abcfs` example from the previous section on setting up logging, use the following procedure to set up logging on a file system that cannot be unmounted. In this example, we'll assume that `/abcfs` cannot be unmounted.

1. **Use the `metainit` command to create the metatrans device:**

```
# metainit d64 -t c0t2d0s6 c2t2d0s1
```

   This creates the metatrans device `/dev/md/dsk/d64`. This metatrans device is composed of the master device `c0t2d0s6` and the logging device `c2t2d0s1`.

2. **Edit the `/etc/vfstab` file to include the new metatrans device.**
   In this case, you would change the line:

```
/dev/dsk/c0t2d0s6   /dev/rdsk/c0t2d0s6   /abcfs   ufs   5   yes   –
```

   to read:

```
/dev/md/dsk/d64   /dev/md/rdsk/d64   /abcfs   ufs   5   yes   –
```

3. **Reboot the system.**
   `fsck` will not notice that `/abcfs` is a logging file system until the system is rebooted again.

Removing Logging from a File System That Cannot be Unmounted

Using the `/abcfs` file system example from above, use the following procedure to remove logging on a file system that cannot be unmounted. Again in this example, we'll assume that `/abcfs` cannot be unmounted.

1. **Use the `metadetach` command to detach the logging device:**

```
# metadetach d64
```

2. **Reboot the system**

3. **Edit the `/etc/vfstab` file to remove the metatrans device.**
   In this case, you would change the line:

```
/dev/md/dsk/d64   /dev/md/rdsk/d64   /abcfs   ufs   5   yes   –
```

   to read:

```
/dev/dsk/c0t2d0s6   /dev/rdsk/c0t2d0s6   /abcfs   ufs   5   yes   –
```

4. **Reboot the system.**

5. **Use the `metaclear` command to clear the metatrans device:**

```
# metaclear d64
```

   This clears the metatrans device `/dev/md/dsk/d64`. Any information pertaining to the master device is rolled from the log prior to clearing the device.

Creating Metatrans Namespace for Exported File Systems

Even if you don't have a spare partition available for a logging device, you can still set up your metatrans devices without a logging device. This is a useful feature if you plan to enable logging on exported file systems, but do not have a spare partition available for the logging device.

To do this, you set up all the metatrans devices and then reboot your clients once. You have to reboot your clients because the device number changes when you convert the master device into a metatrans device. The changed device number results in ESTALE errors being returned to your clients when they attempt to reference files on the converted file system.

The following procedure creates a metatrans device without a logging device. For the purpose of this example, we'll assume the selected file system is `/abcfs`, the master device is `/dev/dsk/c0t2d0s3`, and the metatrans device is `d64`.

1. Unmount the selected file system:

```
# umount /abcfs
```

2. Use the `metainit` command to create the metatrans device:

```
# metainit d64 -t c0t2d0s3
```

3. Edit the `/etc/vfstab` file to include the new metatrans device.
In this case, you would change the line:

```
/dev/dsk/c0t2d0s3   /dev/rdsk/c0t2d0s3   /abcfs   ufs   5   yes   –
```

to read:

```
/dev/md/dsk/d64   /dev/md/rdsk/d64   /abcfs   ufs   5   yes   –
```

4. **Remount the file system:**

```
# mount /abcfs
```

Left as is, the metatrans device you've created has no logging capabilities. You can add a logging device later using the `metattach` command when you have a spare partition.

How DiskSuite Commands Relate to Logging

After you have set up the UFS logging facility on your system, you'll notice some minor differences in how some of the DiskSuite commands function.

- `metaclear` rolls every applicable change from the log and to the master device before clearing the metatrans device.
- `metastat` reports the current status of the metatrans device, the status of the metatrans' master device, and the status of the metatrans' logging device.
- `metainit` creates metatrans devices when the `-t` option is used.
- `metattach` attaches a log to a metatrans device. If the metatrans device is mounted, the log is actually attached when the file system is unmounted or when the system is rebooted.
- `metadetach` removes a log from a metatrans device. If the metatrans device is mounted, the log is actually detached when the file system is unmounted or when the system is rebooted. All of the changes in the log for this master device are rolled forward before the log is detached.

For a complete description and the usage for each of these commands refer to Appendix E, "Using Solstice DiskSuite 4.0 with the SPARCstorage Array 100."

Using Metadevices and Metamirrors

A logging device and/or a master device can be a physical component or a metadevice. Physical components have been used as logging and master devices in the example procedures provided in this chapter. See Chapter 4, “Concatenating and Striping” and Chapter 5, “Mirroring” for examples of how to set up metadevices.

For the sake of reliability and availability, it is strongly recommended that you use metamirrors for logging devices. A device error on the logging device that is a physical component could cause a significant loss of the file system’s data.

Using metadevices for logging or master devices can increase performance and provides you with more configuration options. For example, a striped logging or master device may improve performance. You might want to expand the master device and grow its file system while the metatrans device is still mounted and in use. You could also expand the logging device.

Even if you are unable to mirror the logging and master devices, it is recommended that you configure the logging and master devices as one-way mirrors.

In the example from the section, “How To Set Up UFS Logging” on page 112, in which the logging device was `c2t2d0s1` and the master device was `c0t2d0s6`, we used the `metainit` command as follows:

```
# metainit d64 -t c0t2d0s6 c2t2d0s1
```

to create the metatrans device `/dev/md/dsk/d64`. If the logging device were the metamirror or metadevice `/dev/md/dsk/d12`, you would use the `metainit` command as follows:

```
# metainit d64 -t c0t2d0s6 d12
```

Likewise, if the master device were the metamirror or metadevice `/dev/md/dsk/d5`, the `metainit` command would be:

```
# metainit d64 -t d5 d12
```

See Chapter 5, “Mirroring” for information on how to create metamirrors.

Metatrans Device States

You can determine the state of a metatrans device or a logging device by running the `metastat(1M)` command. The following is a list of the possible states for metatrans and logging devices:

- `Okay` — The device is functioning properly. If mounted, the file system is logging and will not be checked at boot.
- `Hard Error` — A device error or file system panic has occured while the device was in use. An EIO is returned for every read or write until the device is closed or unmounted. The first open causes the device to transition to the `Error` state.
- `Error` — The device can be read and written. The file system can be mounted read-only. However, an EIO is returned for every read or write that actually gets a device error. The device does not transition back to the `Hard Error` state, even when a later device error or file system panic occurs. Successfully completing `fsck` or `newfs` will transition the device into the `Okay` state. When the device is in the `Hard Error` or `Error` state, `fsck` automatically checks and repairs the file system at boot time. `newfs` will destroy whatever data may be on the device.
- `Detached` — The metatrans device does not have a logging device. All benefits from UFS logging are disabled. `fsck` automatically checks the device at boot time. See the manual reference page for `metadetach (1M)` for complete information.
- `Detaching` — The logging device will be detached from the metatrans device when the metatrans device is closed or unmounted. When this occurs, the device transitions to the `Detached` state. See the manual reference page for `metadetach (1M)` for complete information.
- `Attaching` — The logging device will be attached to the metatrans device when the metatrans device is closed or unmounted. When this occurs, the device transitions to the `Okay` state. See the manual reference page for `metattach (1M)` for complete information.

Recovering from Device Errors

Device errors can cause data loss. Read errors occuring on a logging device can cause significant data loss. For this reason, it is strongly recommended that you mirror the logging device.

If a device error occurs on either the master device or the logging device while the metatrans device is processing logged data, the device will transition from the `Okay` state to the `Hard Error` state. If running the `metastat` (1M) command shows that the device is in the `Hard Error` or `Error` state, either a device error has occured or the file system has detected an error in its data and panic'ed. In either case, the recovery process is the same.

To transition the device back into the `Okay` state, use the following steps:

1. **Unmount the affected file system.**
   To determine the affected file system, run the `lockfs` (1M) command. `lockfs` will display the affected file system's lock type as `hard`.

   Every file system sharing the same logging device will be hard locked. You can unmount these file systems even if they were in use when the error occured. If the affected processes try to access an opened file or directory on the hard locked or unmounted file system, an EIO error will be returned.

2. **Backup any accessable data.**
   Before attempting to fix the device error, you may want to recover as much data as possible. If your backup procedure requires a mounted file system (such as `tar` or `cpio`), you can mount the file system read-only. If your backup procedure does not require a mounted file system (such as `dump` or `volcopy`), you can access the metatrans device directly.

3. **Fix the device error.**
   At this point, any attempt to open or mount the metatrans device for read / write access will start rolling all accessable data on the logging device to the appropriate master device(s). Any data that cannot be read or written is discarded. However, if you open or mount the the metatrans device for read-only access, the log is simply rescanned and not rolled forward to the master device(s). In other words, all of the data on the master and logging devices remains unchanged until the first read / write open or mount.

4. **Repair the file system with `fsck (1M)` or `newfs(1M)`.**
   After successfully repairing the file system, `fsck` will automatically transition the file system back to the `Okay` state.

   The `newfs` command will also transition the file system back to the `Okay` state, but will destroy all of the data on the file system. `newfs` is generally used when you plan to restore file systems from backup.

Warning – `newfs` will destroy all existing files and directories currently on the file system.

Note – The `fsck` or `newfs` commands must be run on all of the metatrans devices sharing the same logging device before these devices revert back to the `Okay` state.

Recovering from File System Panics

If a file system detects any internal inconsistencies while it's in use, it will panic the system. If the file system is logging, it will notify the metatrans device that it needs to be checked at reboot. The metatrans device transitions itself to the `Hard Error` state. All other metatrans devices sharing the same logging device also go into the `Hard Error` state.

At reboot, `fsck` checks and repairs the file system and transitions the file system back to the `Okay` state. `fsck` does this for all metatrans devices listed in the `vfstab` file for the affected logging device.

If `fsck` cannot repair one of the file systems, then `fsck` must be run manually on each of the metatrans devices whose file systems were sharing the affected logging device. Only after all of the affected metatrans devices have been checked and successfully repaired will `fsck` reset the state of the errored metatrans device to `Okay`.

Hot Spares 7

The hot spare facility included with DiskSuite allows for automatic replacement of failed submirror and RAID components, provided spare components are available and reserved. Because component replacement and the resyncing of failed components is automatic, hot spares provide additional security from downtime due to hardware failure.

This chapter provides the following information about the use of hot spares with the DiskSuite software package. Use the following table to locate specific information:

Overview of Hot Spares

A hot spare is a component that is running (but not being used) which can be substituted for a broken component in a submirror of a two- or three-way metamirror or RAID device. Failed components in a one-way metamirror cannot be replaced by a hot spare, since no other copy of the data is available.

7

A hot spare is to a metamirror or RAID device what a spare tire is to a car. A spare is meant to quickly replace a flat tire, thus increasing the availability of your car. A spare tire is a temporary fix that is made with the intention that the flat tire will be fixed or replaced with a new one and the spare returned to the trunk.

This is exactly how hot spares should be treated. They are not intended to be used as permanent "fixes" when a component has failed. They are temporary fixes that can be used until a failed component is either fixed or replaced.

Components designated as hot spares cannot be used in submirrors or another metadevice in the `md.tab` file. They must remain ready for immediate use in the event of a component failure.

Hot spares are always in one of three states:

- "Available" hot spares are running and ready to accept data, but are not currently being written to or read from.
- "In-use" hot spares are currently being written to and read from.
- "Broken" hot spares are out of service. A hot spare is placed in the broken state when an I/O error occurs.

You display the states by running the `metahs` command with the `-i` option.

Hot spares can be defined as part of a hot spare pool in the `md.tab` file. This is performed by editing the `md.tab` file to create hot spares and hot spare pools and then running the `metainit(1M)` command.

Hot spare definitions, replacements, additions, and deletions can also be performed by using the `metahs(1M)` utility. The number of hot spare pools is limited to 1000.

Defining Hot Spares

After a hot spare pool is defined, it can be associated with one or more submirrors. Hot spare pools are named `hsp`*nnn* where *nnn* is a number in the range 000-999. A metadevice cannot be configured as a hot spare.

A hot spare can be used in one or more hot spare pools, thus allowing for the maximum amount of security from the minimum number of components. DiskSuite looks for the first available hot spare from the designated pool when errors are reported.

With DiskSuite, you can also define empty hot spare pools so hot spares can be added when they become available.

The procedure for defining hot spare pools is provided below. Note that the hot spares used as examples in these steps are shared in the three defined hot spare pools.

1. **Determine the names of the devices that you are designating as hot spares.**
   In the following steps, the names `/dev/dsk/c0t0d0s2`, `/dev/dsk/c1t0d0s2`, and `/dev/dsk/c2t0d0s2` are used as examples.

2. **Edit the `md.tab` file and set up the hot spare pool(s) using the devices that have been chosen as hot spares.**
   In this example, the hot spare pools are named: `hsp001`, `hsp002`, and `hsp003`. Note that the same hot spares are used in each hot spare pool, but the order is changed.

```
hsp001 /dev/dsk/c0t0d0s2 /dev/dsk/c1t0d0s2 /dev/dsk/c2t0d0s2
hsp002 /dev/dsk/c1t0d0s2 /dev/dsk/c2t0d0s2 /dev/dsk/c0t0d0s2
hsp003 /dev/dsk/c2t0d0s2 /dev/dsk/c0t0d0s2 /dev/dsk/c1t0d0s2
```

3. **Once the hot spare pools are defined, you associate the pools with submirrors using the `metaparam` command with the `-h` option.**

```
# metaparam -h hsp002 d9
# metaparam -h hsp003 d10
# metaparam -h hsp001 d11
```

Once the hot spare pools are defined and associated with a submirror, the hot spares are ‘‘available’’ for use. If a component failure occurs, DiskSuite searches through the list of hot spares in the assigned pool and selects the first ‘‘available’’ component that is equal or greater in disk capacity to the failed component.

If a hot spare of adequate size is found, the hot spare’s state changes to ‘‘in-use’’ and a resync operation is automatically performed. The resync operation brings the hot spare into sync with the other submirrors.

If a component of adequate size is not found in the list of hot spares, the submirror that failed is considered erred and that portion of the submirror no longer replicates the data.

Hot Spare Conditions to Avoid

As stated earlier, hot spares are intended to be a temporary remedy for a failed submirror or RAID component. They are not meant to be used as a permanent replacement. Additionally, there are three conditions that should be avoided when using hot spares:

- Associating hot spares of the wrong size with submirrors. This condition occurs when hot spare pools are defined and associated with a submirror and none of the hot spares in the hot spare pool are equal to or greater than the smallest component in the submirror. This would occur, for example, when 661-Mbyte disk drives make up the hot spare pool that is associated with a submirror that is made up of a 1-Gbyte drive.
- Having all hot spares within the hot spare pool in-use. When the administrator notices that all hot spares are in-use, immediate action is required. There are two possible solutions when this occurs. The first is to add additional hot spares. The second is to repair some of the components that have been hot spare replaced. If all hot spares are in use and a submirror fails due to errors, that portion of the mirror will no longer be replicated.
- Assigning a hot spare pool to a submirror in a one-way metamirror.

Manipulating Hot Spare Pools

DiskSuite offers facilities to dynamically add, delete, replace, and enable hot spares within existing hot spare pools. The command used to perform these functions is `metahs`.

`metahs` is the primary command used to manipulate hot spares and hot spare pools. This command provides the following utilities:

- Adding hot spares to hot spare pools
- Deleting hot spares from hot spare pools
- Replacing hot spares in hot spare pools
- Enabling hot spares
- Checking the status of hot spares

The `metaparam` command is used to add or change the hot spare pool associated with a metamirror or RAID device.

Adding Hot Spares

A hot spare can be added to one or more hot spare pools, using the `metahs` command with the `-a` option. When a hot spare is added, the existing order of the hot spares already in the pool is preserved. The new hot spare is added at the end of list of hot spares in the hot spare pool that is specified.

The following example illustrates adding a hot spare (`/dev/dsk/c0t0d0s7`) to a hot spare pool (`hsp003`).

```
# /usr/opt/SUNWmd/sbin/metahs -a hsp003 /dev/dsk/c0t0d0s7
```

If the hot spare pool, `hsp003`, was defined with two hot spares (for example, `/dev/dsk/c0t1d0s7` and `/dev/dsk/c0t2d0s7`), the hot spare added (`/dev/dsk/c0t0d0s7`) would follow the two that were already associated with that hot spare pool. If the hot spare pool, `hsp003`, didn't already exist, it would be automatically created.

To add a hot spare to the hot spare pools that are currently defined, enter the following:

```
# /usr/opt/SUNWmd/sbin/metahs -a all /dev/dsk/c0t0d0s7
```

The keyword `all` in the above example specifies adding the hot spare, `/dev/dsk/c0t0d0s7`, to all the hot spare pools.

Deleting Hot Spares

Hot spares can be deleted from any or all the hot spare pools to which they have been associated, using the `metahs` command with the `-d` option. DiskSuite will not allow a hot spare to be deleted if it is in the "in-use" state.

When a hot spare is deleted from a hot spare pool, the position of the remaining hot spares changes to reflect the new position. For example, if the second of three hot spares in a hot spare pool is deleted, the third hot spare moves to the second position.

The following example shows the deletion of the hot spare, `/dev/dsk/c0t0d0s7`, from the hot spare pool, `hsp003`.

```
# /usr/opt/SUNWmd/sbin/metahs -d hsp003 /dev/dsk/c0t0d0s7
```

If the hot spare, `/dev/dsk/c0t0d0s7`, is associated with other hot spare pools, it would be removed from all of them if the `all` option is used. For example:

```
# /usr/opt/SUNWmd/sbin/metahs -d all /dev/dsk/c0t0d0s7
```

The `-d` option can also be used to delete a hot spare pool. Before deleting a hot spare pool, all hot spares associated with the hot spare pool must first be deleted. A hot spare pool can not be deleted if it is associated with a submirror.

In the following example, three hot spares (`/dev/dsk/c0t0d0s1`, `/dev/dsk/c1t0d0s1`, and `/dev/dsk/c2t0d0s1`) are associated with hot spare pool `hsp001`. The hot spare pool is currently associated with the metadevice, `d16`. In this example, the metadevice is disassociated with the hot spare pool with the `metaparam` command. Each of the three hot spares are then deleted from the hot spare pool, using `metahs`. Finally, the hot spare pool is deleted.

```
# /usr/opt/SUNWmd/sbin/metaparam -h none d16
# /usr/opt/SUNWmd/sbin/metahs -d hsp001 /dev/dsk/c0t0d0s1
# /usr/opt/SUNWmd/sbin/metahs -d hsp001 /dev/dsk/c1t0d0s1
# /usr/opt/SUNWmd/sbin/metahs -d hsp001 /dev/dsk/c2t0d0s1
# /usr/opt/SUNWmd/sbin/metahs -d hsp001
```

Replacing Hot Spares

Hot spares can be replaced in any or all the hot spare pools to which they have been associated, using the `metahs` command with the `-r` option. However, hot spares that are in the in-use state cannot be replaced by other hot spares.

The order of hot spares in the hot spare pools is not changed when a replacement occurs.

The following example shows the replacement of hot spare `/dev/dsk/c0t0d0s1` with the component `/dev/dsk/c0t1d0s1` in the `hsp003` hot spare pool.

```
# /usr/opt/SUNWmd/sbin/metahs -r hsp003 /dev/dsk/c0t0d0s1 \
                /dev/dsk/c0t1d0s1
```

The `metahs` command with the `-r` command can also be used to replace a hot spare in all the hot spare pools where it is associated. The following example shows the replacement of hot spare `/dev/dsk/c0t0d0s2` with `/dev/dsk/c0t1d0s2`.

```
# /usr/opt/SUNWmd/sbin/metahs -r all /dev/dsk/c0t0d0s2 \
              /dev/dsk/c0t1d0s2
```

The keyword `all` in the above example specifies replacing the hot spare, `/dev/dsk/c0t0d0s2`, with `/dev/dsk/c0t1d0s2` in all the hot spare pools.

Enabling Hot Spares

When a hot spare that has been placed in the “broken” state is repaired, it is brought back to the “available” state by using the `metahs` command with the `-e` option. Hot spares are placed in the “broken” state after an I/O error occurs. After the hardware is repaired, the component can be brought back to the available state.

The following example illustrates placing the hot spare `/dev/dsk/c0t0d0s2` in the available state after it has been repaired.

```
# /usr/opt/SUNWmd/sbin/metahs -e /dev/dsk/c0t0d0s2
```

Changing the Associated Hot Spare Pool

Each submirror or RAID component can optionally be associated with a hot spare pool. The hot spare pool association can be changed with the metaparam command while the system is running, providing none of the hot spares in the current hot spare pool are currently being used by the submirror or RAID component.

In the following example, the hot spare pool, `hsp005`, is currently associated with `d80`. This command changes the hot spare pool association to `hsp001`.

```
# /usr/opt/SUNWmd/sbin/metaparam -h hsp005 d80
```

Checking the Status of Hot Spares

The `metahs` command can be used to display the status of hot spare pools. When the `–i` option is used, `metahs` displays the status for all hot spare pools if one is not specified. For example:

```
# /usr/opt/SUNWmd/sbin/metahs -i
hsp001: 1 hot spare
        /dev/dsk/c0t0d0s0 (Available) blocks 263220
hsp002: 1 hot spare
        /dev/dsk/c0t1d0s0 (In-use) blocks 263220
hsp003: 1 hot spare
        /dev/dsk/c0t2d0s0 (Broken) blocks 263220
```

The above example shows three hot spare pools (`hsp001`, `hsp002`, and `hsp003`) and gives the status of the components. If one of the three hot spare pools had been specified at the command line, only that hot spare pool would have been displayed.

Examples

A series of examples are provided in this section that show, in a step-by-step fashion, how to define hot spares and set up hot spare pools. These examples first define a hot spare pool in the `/etc/opt/SUNWmd/md.tab` file, then add, delete, and replace hot spares in that pool. The examples are:

- Setting up hot spare pools
- Adding hot spares to hot spare pools
- Deleting hot spares from hot spare pools
- Replacing hot spares within hot spare pools

Setting up Hot Spare Pools

Hot spare pools can be defined with or without hot spares. The following steps show how to define the hot spare pool, attach two hot spares, and associate the hot spares with submirrors.

1. **Empty hot spare pools can be defined in the `/etc/opt/SUNWmd/md.tab` file, as follows:**

```
hsp001
hsp002
```

   If you are defining an empty hot spare pool, skip attaching hot spares to the hot spare pool and go to Step 3.

2. **You can attach hot spares to the hot spare pool by entering the following:**

```
hsp001 /dev/dsk/c0t1d0s0
hsp002 /dev/dsk/c0t2d0s0
```

3. **Associate the hot spare pool with a metadevice that will use the hot spares.**
   In this example, the previously defined hot spare pool is associated with both sides of a submirror.

```
d8 -m d9
d9  1 1 /dev/dsk/c0t2d0s7 -h hsp001
d10 1 1 /dev/dsk/c0t3d0s7 -h hsp002
hsp001 /dev/dsk/c0t1d0s0
hsp002 /dev/dsk/c0t2d0s0
```

4. **Use the** `metainit` **command to initialize the two hot spare pools.**
   The submirrors `d9` and `d10` must also be initialized for the hot spares to take affect.

```
# /usr/opt/SUNWmd/sbin/metainit hsp001
# /usr/opt/SUNWmd/sbin/metainit hsp002
# /usr/opt/SUNWmd/sbin/metainit d9
# /usr/opt/SUNWmd/sbin/metainit d10
# /usr/opt/SUNWmd/sbin/metainit d8
# metattach d8 d10
```

Adding Hot Spares to Hot Spare Pools

This section shows how to add one hot spare to the previously defined hot spare pools. In this example, the hot spare `/dev/dsk/c0t3d0s0` is added to both `hsp001` and `hsp002`.

♦ **Use the following command to add the hot spare to both hot spare pools:**

```
# /usr/opt/SUNWmd/sbin/metahs -a all /dev/dsk/c0t3d0s0
```

Using `metahs -i`, the two hot spare pools, `hsp001` and `hsp002`, would appear as follows:

```
# /usr/opt/SUNWmd/sbin/metahs -i
hsp001: 2 hot spares
        /dev/dsk/c0t1d0s0 (Available) blocks 263220
        /dev/dsk/c0t3d0s0 (Available) blocks 263220
hsp002: 2 hot spares
        /dev/dsk/c0t2d0s0 (Available) blocks 263220
        /dev/dsk/c0t3d0s0 (Available) blocks 263220
```

Deleting Hot Spares From Hot Spare Pools

This section shows how to delete a hot spare that was previously added to the two hot spare pools. In this example, the hot spare `/dev/dsk/c0t3d0s0` is being deleted from both `hsp001` and `hsp002`.

♦ **Use the following command to delete the hot spare from both hot spare pools:**

```
# /usr/opt/SUNWmd/sbin/metahs -d all /dev/dsk/c0t3d0s0
```

Using `metahs -i`, the two hot spare pools, `hsp001` and `hsp002`, would appear as follows:

```
# /usr/opt/SUNWmd/sbin/metahs -i
hsp001: 1 hot spare
        /dev/dsk/c0t1d0s0 (Available) blocks 263220
hsp002: 1 hot spare
        /dev/dsk/c0t2d0s0 (Available) blocks 263220
```

Replacing Hot Spares Within Hot Spare Pools

This section shows how to replace a hot spare that was previously associated with a hot spare pool in the `/etc/opt/SUNWmd/md.tab` file. In this example, the hot spare `/dev/dsk/c0t1d0s0` is replaced with `/dev/dsk/c0t3d0s0`.

♦ **Use the following command to replace a hot spare in a hot spare pool:**

```
# /usr/opt/SUNWmd/sbin/metahs -r hsp001 /dev/dsk/c0t1d0s0 \
          /dev/dsk/c0t3d0s0
```

Using `metahs -i`, the hot spare pool `hsp001` would appear as follows:

```
# /usr/opt/SUNWmd/sbin/metahs -i hsp001
hsp001: 1 hot spare
        /dev/dsk/c0t3d0s0 (Available) blocks 263220
```

7

Disksets 8

Solstice DiskSuite's diskset feature lets you set up groups of host machines and disk drives in which all of the hosts in the set are connected to all the drives in the set. These facilities are intended for use in a high-availabilty, fail-over environment, such as the one offered by SunSoft.

Note – While DiskSuite's diskset feature *enables* a high-availability configuration, DiskSuite itself does not actually provide a high-availability environment.

This chapter provides general information about DiskSuite's diskset feature. Specific information for using this feature within a high-availability environment would be contained in the documentation for the particular high-availability product you are using.

Use the following table to locate specific information in this chapter.

8

Overview of Disksets

A *shared diskset* is a grouping of two hosts and disk drives in which all the drives are accessible by both hosts. DiskSuite requires that the device name be identical on each host in the diskset. There is one metadevice state database per shared diskset and one on the local diskset of each host.

Each host in a diskset must have a *local diskset* that is seperate from the shared diskset. A local diskset for a host consists of all drives which are not part of a shared diskset. The host's local metadevice configuration is contained within this local diskset in the local metadevice state database replicas. Refer to Chapter 10, "State Database Replicas," for more information. Only the host knows about its local diskset.

Drives in a shared diskset must not be in any other diskset. None of the partitions on any of the drives in a diskset can be mounted on, swapped on, or part of a local metadevice. Also, all of the drives in a shared diskset must be accessible by both hosts in the diskset.

Figure 8-1 illustrates an example of a diskset shared between two host machines, named *red* and *blue*. The shared diskset is named *relo-red*. Each host's local diskset is shown. Note that this example configuration will be used to describe the procedures in this chapter.

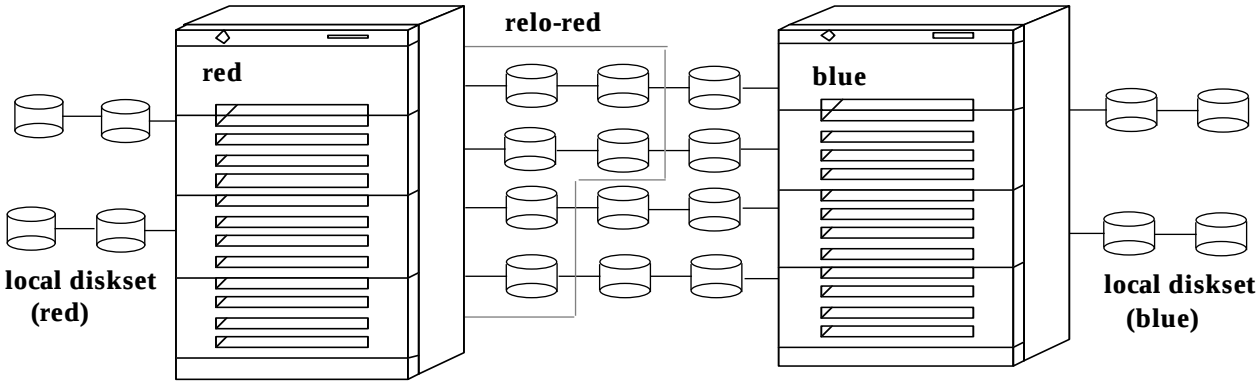


Figure 8-1 Example of a Diskset

Metadevices and hot spare pools in any diskset must consist of drives within that diskset. Likewise, metadevices and hot spare pools in the local diskset must be made up of drives from within the local diskset.

Database Replicas and Disksets

In past releases, DiskSuite has been set up to allow only one metadevice state database per host. The diskset feature actually extends DiskSuite to allow more than one metadevice state database per host. This means that each host in a diskset has a state database for each shared diskset and a seperate state database for its local diskset. This way, if the shared diskset is in use by one host in the set, the other host still can access data from its local diskset.

Naming Conventions

Metadevices within the local diskset use the standard DiskSuite naming conventions. Metadevices within a shared diskset use the following naming conventions:

/dev/md/*setname*/{dsk|rdsk}/d*number*

where *setname* is the name of the diskset, and *number* is the number of the metadevice (usually 0-127).

Hot spare pools within the local diskset use the standard DiskSuite naming conventions. Hot spare pools within a shared diskset use the following naming conventions:

setname/hsp*number*

where *setname* is the name of the diskset, and *number* is the number of the hot spare pool (0 - 999).

DiskSuite Commands and Disksets

The `-s` option is used with the standard DiskSuite commands to create, remove, and administer metadevices and hot spare pools within a specific shared diskset. The `-s` option is available with the following commands:

- `metaclear`
- `metadb`
- `metdetach`
- `metahs`
- `metainit`
- `metaoffline`
- `metaonline`
- `metaparam`
- `metareplace`
- `metastat`
- `metasync`
- `metattach`

Using the `-s` option specifies the diskset that the command will administer. If the `-s` option is not used, the command affects only the local diskset.

Defining Disksets

Before you can begin to create or administer disksets, the DiskSuite 4.0 software must be installed on each host in the diskset and each host must have local database replicas set up. In addition, all disks that you plan to share between hosts in the diskset must be connected to each host and must have the same name on each host.

This section describes how to define disksets from the command line using the `metaset` command with the `-a` option. Additional step-by-step procedures for administering disksets are provided later in this chapter.

There are two basic operations involved in defining disksets:

1. Adding hosts (*adding the first host defines the diskset*)

2. Adding drives

The host names and diskset name used in the following procedures are from the example illustrated in Figure 8-1.

To define a diskset, you begin by naming the two hosts connected to the shared disks:

1. Enter the following on the command line:

```
# metaset -s relo-red -a -h red blue
```

where `relo-red` is the name of the diskset and `red` and `blue` are the names of the first and second hosts you are adding to the set. (The hostname is the same name found in `/etc/nodename`.)

Adding the first host creates the diskset. It is okay if you create a diskset with just one host to start with, and then add the second host at a later time. The last host cannot be deleted until all of the drives within the set have been deleted.

Note – A host name is not accepted if all drives within the diskset cannot be found on each specified host. In addition, a drive is not accepted if it cannot be found on all the hosts in the diskset.

You can look at the status of the new diskset using the `metaset` command with no arguments:

```
# metaset

Set name = relo-red, Set number = 1

Host                  Owner
  red
  blue
```

2. Next, you add drives to the diskset by entering the following:

```
# metaset -s relo-red -a c2t0d0 c2t1d0 c2t2d0 c2t3d0 c2t4d0 c2t5d0
```

where `relo-red` is the name of the previously named diskset and `c2t0d0`, `c2t1d0`, `c2t2d0`, `c2t3d0`, `c2t4d0`, and `c2t5d0` are the names of the drives you are adding to the set.

Note – These are drives names. There is no slice indentifier ("s*x*") at the end of the names.

For a drive to be accepted into a diskset, the drive must not be in use within another metadevice. It must not be currently mounted or swapped on. When a drive is accepted into the diskset, it is repartitioned and the metadevice state database replica for the diskset can be placed on the drive.

Note – A drive name is not accepted if it cannot be found on all hosts specified as part of the diskset.

Once again, you can check the status of the diskset as follows:

```
# metaset

Set name = relo-red, Set number = 1

Host                Owner
  red                Yes
  blue

Drive               Dbase
  c2t0d0             Yes
  c2t1d0             Yes
  c2t2d0             Yes
  c2t3d0             Yes
  c2t4d0             Yes
  c2t5d0             Yes
```

The `metaset` command shows that the 6 drives have been added to the diskset. In addition, the current host `red` is shown as the owner of the diskset. This means the drives in the diskset have been “reserved” for host `red`’s exclusive use.

Note – The first host to add a drive to a diskset becomes the implicit owner of the diskset.

Drives are repartitioned when they are added to a diskset only if Slice 7 is not set up correctly. A small portion of each drive is reserved in Slice 7 for use by DiskSuite. The remainder of the space on each drive is placed into Slice 0. Any existing data on the disks will be lost by the repartitioning. After adding a drive to a diskset, your system administrator can repartition the drive using `fmthard(1M)` as necessary, with the exception that Slice 7 is not altered (moved, resized, etc.) in any way.

If a database has been placed automatically on a drive (in Slice 7), `metaset` indicates this with a “Yes” in the `Dbase` column.

Unlike local metadevice administration, it is not necessary to create or delete state database replicas manually. The DiskSuite software tries to balance a reasonable number of replicas across all drives in a diskset.

Administering Disksets

After drives are added to a diskset, the diskset can be *reserved* (or *taken*) and *released* by hosts in the diskset. When a diskset is reserved by a host, the other host in the diskset cannot access the data on the drives in the diskset. In order to perform maintenance on a diskset, a host must be the owner of the diskset or have reserved the diskset. In the example shown previously, host *red* took implicit ownership of the diskset by putting the first drives into the set.

The SCSI `reserve` command is issued to each drive in the diskset to reserve it for exclusive use by the current host. Each drive in the diskset is probed once every second to determine that it is still reserved.

Note – If a drive has been determined unexpectedly not to be reserved, the host will panic. This behavior helps to minimize data loss which would occur if two hosts were to simultaneously access the same drive.

Reserving a Diskset

Before a host can use drives in a diskset, the host must reserve the diskset. There are two methods of reserving a diskset:

- *Safely* - When you safely reserve a diskset, `metaset` checks to see if another host currently has the set reserved. If another host has the diskset reserved, your host will not be allowed to reserve the set.
- *Forcibly* - When you forcibly reserve a diskset, `metaset` reserves the diskset whether or not another host currently has the set reserved. This method is generally used when a host in the diskset is down or not communicating. All disks within the set are taken over and FailFast is enabled. The metadevice state database is read in and the shared metadevices configured in the set become accessible. If the other host had the diskset reserved at this point, it would panic due to reservation loss.

Normally, two hosts in a diskset cooperate with each other to ensure that drives in a diskset are reserved by only one host at a time. A normal situation is defined as both hosts up and communicating with each other.

♦ **To safely reserve a diskset, enter the following:**

```
# metaset -s relo-red -t
```

where `relo-red` is the name of the diskset.

In this case, host `red` communicates with host `blue` and ensures that host `blue` has released any reservation of the diskset before host `red` attempts to reserve the set.

In some cases, a host in the diskset may be down or not communicating. In order to reserve a diskset under these circumstances, the diskset must be forcibly reserved.

♦ **To forcibly reserve a diskset, enter the following:**

```
# metaset -s relo-red -t -f
```

where `relo-red` is the name of the diskset.

In this case, host `red` does not communicate with host `blue`. Instead, the drives in the diskset are reserved without warning. If host `blue` had the diskset reserved, it would now panic due to reservation loss.

Note – If you are fairly certain that the hosts in the diskset are communicating, it is normally a good idea to perform a safe reservation.

Releasing a Diskset

Sometimes it may be desirable to release a diskset. Releasing a diskset can be useful when performing maintenance on the drives in the set. When a diskset is released, it cannot be accessed by the host. If both hosts in a diskset release the set, neither host in the diskset can access the drives in the set.

You can release a diskset by entering the following:

```
# metaset -s relo-red -r
```

where `relo-red` is the name of the diskset you are releasing.

You can verify that the diskset has been released on this host by using the `metaset` command with no arguments as follows:

```
# metaset

Set name = relo-red, Set number = 1

Host                Owner
  red
  blue

Drive               Dbase
  c2t0d0             Yes
  c2t1d0             Yes
  c2t2d0             Yes
  c2t3d0             Yes
  c2t4d0             Yes
  c2t5d0             Yes
```

Note that there is no owner of the diskset. Viewing status from host `red` could be misleading. A host can only determine if it does or does not own a diskset. For example, if host `blue` were to reserve the diskset, it would not appear so from host `red`; though it would from host `blue`.

Removing Hosts and Drives From a Diskset

After a drive is added to a diskset, it can be removed using the `metaset` command with the `-d` option. For example, to remove `c2t5d0` from diskset `relo-red`, you would type the following:

```
# metaset -s relo-red -d c2t5d0
```

When drives are removed from a diskset, DiskSuite re-balances the metadevice state database replicas across the remaining drives.

Note – The `-f` option must be specified when you are deleting the last drive in a set, since this drive would implicitly contain the last state database replica.

You can also remove hosts from a diskset. The last host can be removed from a diskset only after all drives in the diskset have been removed. Removing the last host from a diskset destroys the diskset.

For example, to remove host `blue` from the diskset `relo-red`, you would type the following:

```
# metaset -s relo-red -d -h blue
```

You can verify that host `blue` has been removed from the diskset using the `metaset` command as follows:

```
# metaset

Set name = relo-red, Set number = 1

Host                Owner
  red                Yes

Drive               Dbase
  c2t0d0             Yes
  c2t1d0             Yes
  c2t2d0             Yes
  c2t3d0             Yes
  c2t4d0             Yes
  c2t5d0             Yes
```

As you can see, host `blue` is no longer part of the diskset.

Adding Drives or Hosts to an Existing Diskset

Drives can be added to a diskset after the diskset has been defined. To add a drive to an existing diskset, enter the following:

```
# metaset -s relo-red -a c2t5d0
```

where `relo-red` is the name of the diskset and `c2t5d0` is the drive being added.

When drives are added to a diskset, DiskSuite re-balances the metadevice state database replicas across the remaining drives.

Note – When drives are added to a diskset, they are repartitioned as described in “Defining Disksets” on page 141.

You can also add a host to an existing diskset. With DiskSuite, a maximum of two hosts per diskset are supported. For example, to add host `blue` back to the diskset `relo-red`, you would enter the following:

```
# metaset -s relo-red -a -h blue
```

You can verify that host `blue` has been added to the diskset using the `metaset` command as follows:

```
# metaset

Set name = relo-red, Set number = 1

Host              Owner
  red              Yes
  blue

Drive             Dbase
  c2t0d0           Yes
  c2t1d0           Yes
  c2t2d0           Yes
  c2t3d0           Yes
  c2t4d0           Yes
  c2t5d0           Yes
```

8

RAID Devices 9

This chapter discusses RAID (Redundant Arrays of Inexpensive Disks) device configuration information. The information provided includes hardware and software considerations along with instructions for managing RAID devices. Use the following table to locate specific information.

RAID Overview

DiskSuite RAID devices support RAID Level 5. RAID Level 5 configuration allows you to recover from a single disk failure. It can also be more cost effective than mirroring disks, especially when it comes to hardware costs.

RAID must be comprised of three or more physical partitions. Each partition is referred to as a component. A RAID metadevice can be grown by concatenating additional partitions to the metadevice.

RAID level 5 includes multiple physical partitions used to simulate a single large slice (partition). A single sector on one of these physical slices contain either a sector's worth of data, or parity information relating to the data on the same sector of all other slices in the array.

In order to eliminate a parity partition as a bottleneck, no one physical partition will hold all of the parity information; it will be placed on different partitions for different sectors.

Warning – Because RAID devices require the parity to be intermingled with data, running `metainit` on devices that comprise a RAID device destroys existing data.

Operation of RAID

The following operations are supported for RAID metadevices using Soltice DiskSuite 4.0:

- Configuring and defining RAID metadevices.
- Concatenating new components to existing RAID metadevices.
- Allocating hot spare pools to RAID metadevices to provide component backup in the event of original component failures.
- Replacing errored components or in-use hot spare components with a new component.
- Resynchronizing components during reboot.
- Viewing the status of RAID metadevices.
- Clearing RAID metadevices.

Each of these operations is discussed in more detail in the following sections.

Creating RAID Metadevices

In defining a RAID metadevice, the definition of the metadevice is included on the command line as options and parameters to the `metainit` command. The following is an example of a RAID metadevice command line definition:

```
# metainit /dev/md/dsk/d80 -r /dev/dsk/c0t0d0s1 \
/dev/dsk/c1t0d0s1 /dev/dsk/c2t0d0s1 -i 8k
```

The option parameters specified in the example above define the characteristics of the RAID metadevice. The `-r` option informs the `metainit` utility that the metadevice being defined is a RAID metadevice. The `-i` option defines the interlace size (8 kilobytes) that is to be used when configuring the RAID metadevice for striping of the data and parity regions of the components defined on the command line, `/dev/dsk/c0t0d0s1`, `/dev/dsk/c1t0d0s1`, and `/dev/dsk/c2t0d0s1` in the example above.

The following sections describe how to check the status of RAID metadevices and how to alter their configuration. Step-by-step examples demonstrate the entire procedure for setting up RAID metadevices.

Resyncing RAID Devices

All RAID metadevices are resynchronized during reboot via the `metasync -r (1M)`command. During the resync operation, the state of the RAID metadevice is validated and any operations that may have been halted due to a system panic, a system reboot, or a failure to complete (possibly due to an I/O error on an individual component) are restarted. Any of the states defined in “Checking Status” on page 156, are valid unless components are found to be either in the Initializing or Resyncing state and the overall RAID metadevice state contradicts this information. Upon validation of a RAID metadevice, if the state of any components of the metadevice are in the Initializing or Resyncing state, the appropriate operation (init or resync, respectively) will be reinitiated on the component(s) necessary.

Once the `metasync` operation completes, the state of the metadevice may either be in the “Okay” state, indicating full component availability, or in one of the two maintenance states (“Maintenance” or “Last Erred”), indicating

errors were encountered on component(s) during the resync operation. If one of the latter states prevails, use the `metareplace(1M)` command to perform the appropriate level of data recovery.

Reconfiguring RAID Devices

Reconfiguration of a RAID metadevice means altering the original composition of the metadevice either by concatenating additional components, replacing components (error or non-errored), or assigning a hot spare pool to the metadevice (to provide a backup in the event that errors are encountered on any of the components of the metadevice). Each of these reconfiguration options is discussed in more detail in the following sections.

Concatenating Components

Concatenation is the appending of new components to an existing metadevice. Concatenation of a component to a RAID metadevice allows the device to grow by allocating additional disk space from the concatenated components.

Note – No parity information is stored on the newly appended components.

The following steps "grow" an existing metadevice by adding a single concatenated component:

1. Use `metattach(1M)` specifying the RAID metadevice to grow and the new component(s) to add to the metadevice configuration.
2. If a UFS filesystem exists on the RAID metadevice, run `growfs(1M)` to update the filesystem so that the additional space is allocated and recognized by the filesystem.

At this point the new component has been attached to the metadevice and may be used as though it were an original component.

Note – Once a component is attached to a RAID device, it cannot be removed.

Replacing Components

Once an I/O error is detected on a component of a RAID metadevice, no further I/O's will be performed on that component (unless the component is in the "Last Erred" state, refer to "Checking Status" on page 156). In this situation, the administrator would most likely want to perform some type of error recovery such that the state of the RAID device is non-errored and the possibility of data loss is reduced.

There are two methods for performing recovery:

- attempt to recover from possible soft errors by enabling the currently errored component
- replace the existing errored component with a new component

Both of these methods involve the use of the `metareplace(1M)` utility to perform component replacement and data recovery. During component replacement, data is recovered. If a hot spare is currently in use, the data is copied from the hot spare. When no hot spare is in use, data is rebuilt using the parity.

The following steps demonstrate how to replace an errored component of a RAID metadevice in which only one component is errored.

1. Determine if you have any additional components that are available to be used to replace the errored component.
2. If other device components are available, run `metareplace` with the new component.
3. If no other components are available, run `metareplace` with the `-e` option to attempt to recover from possible soft errors by resyncing the errored device.

If multiple errors exist, the device in the "Maintenance" state must first be replaced or enabled.

Caution – Replacing an errored component when multiple components are in error may cause data to be fabricated. The integrity of the data in this instance is questionable.

Note that you can use the `metareplace` command on non-errored devices to change a disk. This can be useful for tuning performance of RAID devices.

Changing Hot Spare Pool Association

The only parameter of a RAID metadevice that may be altered is the allocation of hot spare pools. The hot spare pool association can be changed on an existing RAID metadevice regardless of whether the metadevice is in use or not. Hot spare pools may be allocated, deallocated, or reassigned at anytime unless a component in the hot spare pool is currently being used to replace an errored component in the RAID metadevice.

The following steps describe how to allocate a hot spare pool to a RAID metadevice:

1. Create and configure a RAID metadevice.
2. Create a hot spare pool of unused disk partitions
3. Run the `metaparam(1M)` utility to assign this hot spare pool to an existing RAID metadevice.

Once the hot spare pool has been assigned to the metadevice, any components that are currently errored or error in the future will be replaced by a component in the hot spare pool as long as hot spare components are available and are at least as large as the smallest component in the metadevice.

Note – To avoid data fabrication, DiskSuite will not allow hot sparing of a metadevice if any devices within that metadevice are in the “Last Erred” state.

Checking Status

Like other metadevices --mirrors, stripes and concatenations-- the status of RAID metadevices may be observed using the `metastat (1M)` command. The states of RAID metadevices vary as do the components of RAID metadevices. The following are brief descriptions of the possible states of RAID metadevices.

- *Initializing* - This state indicates that the components are in the process of having all disk blocks zeroed. This is necessary due to the nature of RAID metadevices with respect to data and parity interlace striping. If an I/O

error occurs during this process, the device will go into the “Maintenance” state. If the initialization fails, the metadevice is in the “Init Failed” state and the component is in the “Maintenance” state.

To recover from this condition, run `metaclear(1M)` to clear the RAID metadevice and reinitialize the RAID metadevice with a new component to replace the one in error.

Once the state of the RAID metadevice changes to the “Okay” state, the initialization process is complete and you are once again able to open to RAID device. Up to this point, applications will continue getting the error message.

- *Okay* - This state indicates that the RAID metadevice is ready for use and is currently error free. In this state, components may be added (`metattach(1M)`) or replaced (`metareplace(1M)`).
- *Maintenance* - This state indicates that a single component has been marked as errored due to I/O or open errors encountered during a read or a write operation on the component. If this situation occurs, an `invoke` recovery message is displayed which helps the administrator determine the appropriate action to clear this error condition. For RAID metadevices, the only appropriate corrective action is to `metareplace(1M)` the errored component with a new component. Once the replacement completes successfully, the new component will be displayed and the state of the metadevice and the component will be “Okay”.
- *Maintenance/Last Erred* - This state indicates that multiple components in the RAID metadevice have encountered errors. The state of the errored components will be marked either “Maintenance” or “Last Erred”. In this state, no I/O will be attempted on the component that is in the “Maintenance” state, but I/O will be attempted to the component marked “Last Erred” with the outcome being the overall status of I/O request. Once in this state, recovery may still be accomplished by invoking the `metareplace` command as described on the `invoke` line of the status output. The `metareplace` command must be issued with the `-f` flag; this indicates that the administrator knows that data may be fabricated due to the multiple errored components.

  Note that if a device is in the “Last Erred” state and an attempt is made to replace the device, data may be fabricated.

9

Hardware and Software Considerations

There are both hardware and software considerations that affect RAID metadevices.

The software considerations include:

- The values assigned to the interlace size when building a RAID metadevice
- Concatenation
- Performance
- The use of a RAID metadevice as a component of another metadevice

The hardware considerations include:

- Mixing different size components
- The number of controllers
- Mixing components with different geometry
- The I/O load on the bus

Assigning Interlace Values

The key to performance using RAID is the interlace value. The value is user configurable at the time a metadevice is created. Thereafter, the value cannot be modified.

The interlace value defaults to 16Kbytes. This is a reasonable value for most applications. If the different components in the RAID metadevice reside on different controllers and the accesses to the metadevice are primarily large sequential accesses, then a interlace value of 32Kbytes may have better performance.

Concatenating to a Device

Concatenating a new component to an existing RAID metadevice will have an impact on the overall performance of the metadevice because the data on concatenated components is sequential; data is not striped across all components. The original components of the metadevice, as discussed in the introduction, have data and parity striped across all components. This striping is lost for the concatenated component, although the data will still be recoverable if this component errors since the parity will still be used during component I/O.

Concatenated components also differ in the sense that they will not have parity striped on any of the regions, thereby, allowing the entire contents of the component (up to the current component size) to be available for data.

Any performance enhancements for large or sequential writes are lost when components are concatenated.

Write Performance

A RAID metadevice maintains parity for multiple partitions. When a write is issued to a RAID metadevice, multiple I/Os are performed to adjust the parity. For this reason, the type of application should be considered; applications with a high ratio of reads to writes will perform better on a RAID device.

Performance of a Degraded Device

When a RAID metadevice requires maintenance due to a failed disk, the parity is used to reconstruct the data; this requires reading multiple partitions to reconstruct the data. The more components assigned to a RAID metadevice with a failed disk, the longer a read or write operation will take. This applies to resyncing the metadevice as well as normal I/O activity.

RAID as a Component to a Device

A RAID metadevice cannot be used as a submirror or as a component to a concatenation or stripe. A RAID metadevice may be used as either the master or log device of a metatrans device.

Mixing Different Size Components

When different size disk components are used in a RAID metadevice, some disk space will be unused unless the unused portion is assigned to another metadevice. This is because the metadevice is limited by the smallest partition in the configuration (*n* times the smallest component, where *n* is the number of components in the metadevice). For example, if there are two 327 Mbyte partitions and one 661 Mbyte partition in a RAID metadevice, the metadevice will only use 327 Mbytes of the space on the 661 Mbyte partition.

To assign the unused disk space to another metadevice, the component must be repartitioned (use `format`(1M)).

Note – You should never repartition a component which is in a RAID device.

Using Components with Different Geometry

All components in a RAID metadevice should have the same number of sectors and tracks per cylinder. This is referred to as the disk geometry. The geometry is related to the capacity of the drive. Disk geometry varies depending on the manufacturer and type.

The problem with the differing component geometries is that the UFS file system attempts to lay out file blocks in an efficient manner. UNIX counts on a knowledge of the component geometry and uses cylinder groups to attempt to minimize seek distances. If all components do not have the same geometry, the geometry of the first component is reported to the file system. This may cause the efficiency of the file system to suffer.

Controllers

Building a RAID metadevice with all the component partitions on the same controller will adversely affect I/O performance. Also, creating a metadevice of components on different controller types can affect performance because some controllers are faster than others. The I/O throughput will be limited by the slowest controller.

An example of a controller limiting performance is when several devices (e.g., 3 Mbyte per second disks) are attached to the same controller. In this instance, the throughput may be limited to the throughput of the controller and not the sum of the devices.

Another factor to consider when configuring RAID metadevices with respect to controllers is the possibility of the controller being the single-point-of-failure. RAID provides the capability to recover data when a single errored component exists within the configured metadevice, but when multiple components are errored (that is, controller failure) the task of data recovery may or may not succeed.

Examples

Examples of the basic RAID operations are defined in this section.

The examples include:

- Configuring a RAID metadevice and monitoring its status during initialization
- Concatenating a new component to an existing RAID metadevice
- Replacing an errored component and monitoring the progress of the component resync process

Defining a RAID device

The following example shows how to define a RAID metadevice of four components with a interlace size of 10 Megabytes.

In the following example, the four components comprising the RAID metadevice are `/dev/dsk/c1t0d0s2`, `/dev/dsk/c2t0d0s2`, `/dev/dsk/c3t0d0s2`, and `/dev/dsk/c4t0d0s2`. The RAID metadevice will be identified by `d10`.

1. **Verify that the components and RAID definition are valid using `metainit -n`.**

   The `-n` option validates of the command line syntax without performing actual metadevice initialization.

```
# /usr/opt/SUNWmd/sbin/metainit -n d10 -r /dev/dsk/c1t0d0s2 \
/dev/dsk/c2t0d0s2 /dev/dsk/c3t0d0s2 /dev/dsk/c4t0d0s2 -i 10m
```

2. **If the configuration is accurate, run `metainit` to initialize the RAID metadevice.**

```
# /usr/opt/SUNWmd/sbin/metainit d10 -r /dev/dsk/c1t0d0s2 \
/dev/dsk/c2t0d0s2 /dev/dsk/c3t0d0s2 /dev/dsk/c4t0d0s2 -i 10m
```

3. **While the metadevice is initializing, you can use** `metastat` **to view the progress.**
   The RAID metadevice will not be available for use until the completion of the initialization cycle. At this point the state of the metadevice will transition from “Initializing” to “Okay”.

```
# metastat d10
d80: RAID
     State          : Initializing
Initialization in progress: 16% done
     Size           : 16608 blocks
Original device:
     Size           : 16608 blocks
Device              Start Block  Dbase      State    Hot Spare
c1t0d0s2                  471      No     Initializing
c2t0d0s2                  170      No     Initializing
c3t0d0s2                  170      No     Initializing
c4t0d0s2                  170      No     Initializing

# metastat d10
d80: RAID
     State          : Okay
     Size           : 16608 blocks
Original device:
     Size           : 16608 blocks
Device              Start Block  Dbase      State    Hot Spare
c1t0d0s2                  471      No        Okay
c2t0d0s2                  170      No        Okay
c3t0d0s2                  170      No        Okay
c4t0d0s2                  170      No        Okay
```

Concatenating to a RAID Device

The following example shows how to concatenate a new component to an existing RAID metadevice on which a file system exists.

In the following example, the RAID metadevice is `d10`, and the new component to attach is `/dev/dsk/c5t0d0s2`.

1. **Use `metastat` to check the status of the RAID metadevice to which the new component will be attached.**

```
# metastat d10
d80: RAID
     State           : Okay
     Size            : 16608 blocks
Original device:
     Size            : 16608 blocks
Device            Start Block    Dbase       State     Hot Spare
c1t0d0s2                471        No          Okay
c2t0d0s2                170        No          Okay
c3t0d0s2                170        No          Okay
c4t0d0s2                170        No          Okay
```

2. **If the state of the RAID configuration is stable, use `metattach` to attach the new component to this metadevice.**

```
# metattach d10 /dev/dsk/c5t0d0s2
```

3. **While the new component is initializing, you can use `metastat` to view the progress.**
   The new RAID component will not be available for use until the completion of the initialization cycle. At this point the state of the component will transition from "Initializing" to "Okay".

```
# metastat d10
d80: RAID
     State           : Okay
     Size            : 27680 blocks
Original device:
     Size            : 16608 blocks
Device              Start Block  Dbase      State    Hot Spare
c1t1d0s2                 471       No          Okay
c2t0d0s2                 170       No          Okay
c3t0d0s2                 170       No          Okay
c4t0d0s2                 170       No          Okay
Concatenated Devices:
     Size            : 11072 blocks
Device              Start Block  Dbase      State    Hot Spare
c0t2d0s0                 935       No          Okay
```

4. **Use the `growfs` command to expand the mounted file system.**
   `growfs` is a non-destructive utility and may be issued while the file system is mounted.

```
# growfs /dev/md/rdsk/d10 -M /foo
```

   where `/foo` is the mount point.

Recovering from Component Errors

The following example shows how to recover when a single component in a RAID metadevice is errored.

In the following example, the RAID metadevice is `d10`, and the component that will be used to replace the errored component is `/dev/dsk/c5t0d0s2`.

1. **Using** `metastat`**, identify the component that is errored and needs to be replaced.**
   The state of the metadevice and the errored component will be "Maintenance". When in this state, a line will be displayed with the action that should be taken to recover from this state.

```
# metastat d10
d80: RAID
     State           : Maintenance
Invoke         :metareplace d10 /dev/dsk/c0t2d0s5 <new device>"
     Size            : 16608 blocks
Original device:
     Size            : 16608 blocks
Device              Start Block  Dbase     State     Hot Spare
c1t0d0s2                 471       No        Okay
c2t0d0s2                 170       No        Okay
c3t0d0s2                 170       No        Maintenance
c4t0d0s2                 170       No        Okay
```

2. **Use the** `metareplace` **command to perform the replacement of the errored component as follows.**

```
# metareplace d10 c3t0d0s2 c5t0d0s2
```

3. Use `metastat` to monitor the progress of the replacement.
During the replacement of the errored component the state of the metadevice and the new component will be “Resyncing”. While in this state, you may continue the metadevice.

```
# metastat d10
d80: RAID
     State          : Resyncing
Resync in process    : 21% done
     Size           : 16608 blocks
Original device:
     Size           : 16608 blocks
Device              Start Block  Dbase     State     Hot Spare
c1t0d0s2                 471       No        Okay
c2t0d0s2                 170       No        Okay
c5t0d0s2                 170       No        Resyncing
c4t0d0s2                 170       No        Okay
```

State Database Replicas 10

The DiskSuite state database replicas are dedicated portions of a disk, similar to a disk label. The space occupied by the replica is reserved for the exclusive use of the metadevice state database; it cannot be used for any other purpose.

State database replicas are critical to the operation of all metadevices because they provide a memory service for DiskSuite. The replicas keep track of configuration and status information for all metadevices, metamirrors, metatrans devices, and hot spares. The replicas also keep track of error conditions that have occurred.

The replicated state database information keeps DiskSuite operating. Without replicas (copies) of the same information for comparison, DiskSuite does not know the current running state of metadevices. This chapter and the `metadb(1M)` manual page provide a detailed discussion of how the replicas are used by the metadisk driver.

Each replica can exist on either a dedicated disk partition or within space specifically reserved for a replica within a striped or concatenated metadevice, or a logging device. You can store multiple replicas in a single disk partition, however, placing multiple replicas on a single disk reduces reliability. Each replica occupies 517 Kbytes or 1034 disk blocks of the partition.

The state database must be initialized before any metadevices are configured. See Chapter 2, “Installation and Setup,” for information about setting up the initial state database.

10

This chapter provides information on how to use the state database and its associated replicas with the DiskSuite software package. Use the following table to locate specific information.

Overview of the State Database Replicas

After you have configured metadevices, the metadevice driver must "remember" this configuration and status information. The metadevice state database is the metadevice driver's long-term memory. The metadevice driver stores all the metadevice configuration information in the state database. This includes the configuration information about metadevices, metamirrors, metatrans devices, and hot spares. This is possibly the same information that exists in the `/etc/opt/SUNWmd/md.tab` and `/etc/opt/SUNWmd/md.cf` files, but may differ if the `/etc/opt/SUNWmd/md.tab` file has not been kept up to date or if failed submirror components have been replaced with hot spares.

If the replicated metadevice state database were lost, the metadevice driver would have no way of knowing any configuration information. This could result in the loss of all data stored on metadevices. To protect against losing the metadevice state database because of hardware failures, multiple replicas (copies) of the state database are kept.

These multiple replicas also protect the state database against corruption that can result from a system crash. Each replica of the state database contains a checksum. When the state database is updated, each replica is modified, one at a time. If a crash occurs while the database is being updated, only one of the replicas will be corrupted. When the system reboots, the metadevice driver uses the checksum embedded in the replicas to determine if a replica has been corrupted. Any replicas that have been corrupted are ignored.

If a disk containing the metadevice state database is turned off, the metadevices remain fully functional because the database is retrieved from one of the replicas still in operation. Changes made to the configuration following the reboot are stored only in those replicas that are running when the system comes back up. If the disk drive that was turned off is later turned back on, the data contained in the replica stored on that disk will be ignored.

Basic State Database Operation

The locator block contains the location and status of all known replicas. The primary information contained in the status is whether a particular replica is up to date. A single commit counter is also contained in the locator block. This commit counter is incremented every time the status of any replica changes. The metadevice driver uses this commit counter to select replicas that have the latest locator block. It then uses the data contained in a replica that is both up to date and has not been corrupted by a system crash.

The following are the basic steps a system goes through when the metadevice state database is accessed:

1. The locator block is read from all database replicas that have been patched into `/etc/system` or passed in from `/etc/opt/SUNWmd/mddb.cf`.
2. The locator block is read from any replicas mentioned in previously read locator blocks but not patched into `/etc/system` or passed in from `/etc/opt/SUNWmd/mddb.cf` by `metainit`.
3. One of the locator blocks with a commit counter equal to the highest commit counter is used to select which replicas are up to date.
4. One of the up-to-date replicas is read.
5. If the checksum is correct, the state database is set up.
6. If the checksum is not correct, another of the up-to-date replicas is read and tested for the correctness of the checksum. This continues until a correct replica is found.

If there are no more replicas to test for correctness, DiskSuite gives up and an error message is displayed stating that a usable replica cannot be found. If no replicas can be found that pass the checksum test, the database has been lost and you must start over building the initial state database and redefining all metadevices.

If half or more of the replicas referenced in the locator block were not valid, the metadevices are marked read only. One cause of this situation could be that a component was accidentally turned off or disconnected. If this is the case, you must turn on drives that contain other replicas and reboot the system. The replicas located on these components will be found and checked when the system reboots.

It is also possible that half or more of the replicas were lost due to hardware failure. For instance, if there were five replicas and three were lost, DiskSuite will report that the system cannot boot. The references to the three other replicas can be deleted (using the `metadb -d` command), and the two remaining replicas would be updated to say they are the only two replicas. The system would then boot.

Planning Locations of Replicas

Planning the location of replicas on a system where DiskSuite has just been loaded involves several considerations. These considerations include:

- Database replicas can reside on any unused partition or on any partition which will also be part of a metadevice or logging device with the exception of `root`, `swap`, `/usr`, or an existing file system.
- If multiple controllers exist, replicas should be spread as evenly as possible across the controllers.
- If multiple disks exist on a controller, at least two of the disks on each controller should store a replica of the metadevice state database.
- At least three replicas should be created. For instance, if you have three components, a replica should be created on each. That way, if one component fails, you will have the necessary two replicas to continue running. When you have less than two replicas, DiskSuite will not function.

Note – In a two-component configuration, you should always create two replicas on each component. For example, assume you create two replicas on one component and only one replica on the other. If the component with two replicas fails, DiskSuite will not function because the remaining component only has one replica.

- No more than one replica should be placed on a single disk unless that is the only way to reach the minimum number (three) of replicas.

Creating a State Database

You create the initial state database by using the `metadb` command with both the `-a` and `-f` options, followed by the device name where the replica is to reside. For example:

```
# /usr/opt/SUNWmd/sbin/metadb -a -f /dev/dsk/c0t1d0s0
```

The `-a` option specifies that a replica (in this case, the initial) state database should be created. When the `-a` option is used, the `/etc/system` file is automatically patched with the new information about the state database or replica location and the `/etc/opt/SUNWmd/mddb.cf` file is updated. The `-f` option forces the creation to occur, even though a state database does not exist.

The `-a` and `-f` options should only be used together when no state databases exist.

There must be at least two replicas located on the system at any time. Otherwise, if the system crashes, it is possible all metadevice configuration data may be lost.

Creating Replicas

Additional replicas, containing identical information, are necessary to prevent the loss of the configuration information. Losing this information would make operation of the metadevices impossible.

You can add replicas from the command line or by editing the `/etc/opt/SUNWmd/md.tab` file.

When the `/etc/opt/SUNWmd/md.tab` is edited, a line of the form:

```
mddb01 /dev/dsk/c0t2d0s0
```

is entered, and the `metadb -a` command is run as follows:

```
# /usr/opt/SUNWmd/sbin/metadb -a mddb01
```

To create additional replicas from the command line using the `metadb -a` command, the following would be entered:

```
# /usr/opt/SUNWmd/sbin/metadb -a /dev/dsk/c0t2d0s0
```

In either of the above cases, a replica of the state database is created on `/dev/dsk/c0t2d0s0`.

All replicas that are located on the same partition of a physical device must be created at the same time. Thus, if additional replicas were to be created on `/dev/dsk/c0t2d0s0`, they would need to be created when the first one is defined. For example:

```
# /usr/opt/SUNWmd/sbin/metadb -a -c 2 /dev/dsk/c0t2d0s0
```

The above command would create two replicas on `/dev/dsk/c0t2d0s0`.

Removing Replicas

With DiskSuite, you can remove all replicas of the state database. Although all replicas of the state database can be removed, this should never be done if metadevices are still configured. If all the replicas are removed, all metadevices will become inoperable.

Removal is done from the command line using the `metadb -d` command followed by the name of the replica to be removed. Editing the `/etc/opt/SUNWmd/md.tab` file and removing the entry for a replica does not remove the replica.

For example, if replicas had been set up on `/dev/dsk/c0t2d0s0` and `/dev/dsk/c0t1d0s0`, they could be removed by entering:

```
# /usr/opt/SUNWmd/sbin/metadb -d /dev/dsk/c0t2d0s0 \
                /dev/dsk/c0t1d0s0
```

If the replicas, `/dev/dsk/c0t2d0s0` and `/dev/dsk/c0t1d0s0`, had been defined in the `/etc/opt/SUNWmd/md.tab` file and aliased as `mddb01`, they could also be removed with the following command:

```
# /usr/opt/SUNWmd/sbin/metadb -d mddb01
```

Checking the Status of Replicas

You can use `metadb` to view the status of all replicas by using the `-i` option. An example of output from the `metadb -i` command follows:

```
example# /usr/opt/SUNWmd/sbin/metadb -i
       flags       first blk       block count
    a m  p  luo     16              1034          /dev/dsk/c0t3d0s3
    a    p  luo     1050            1034          /dev/dsk/c0t3d0s3
    a    p  luo     16              1034          /dev/dsk/c1t3d0s3
    a    p  luo     1050            1034          /dev/dsk/c1t3d0s3
    a    p  luo     16              1034          /dev/dsk/c0t2d0s3
    a    p  luo     1050            1034          /dev/dsk/c0t2d0s3
o - replica active prior to last mddb configuration change
u - replica is up to date
l - locator for this replica was read successfully
c - replica's location was in /etc/opt/SUNWmd/mddb.cf
p - replica's location was patched in kernel
m - replica is master, this is replica selected as input
W - replica has device write errors
a - replica is active, commits are occurring to this replica
M - replica had problem with master blocks
D - replica had problem with data blocks
F - replica had format problems
S - replica is to small to hold current data base
R - replica had device read errors
example#
```

In the example above, the characters in front of the device name represent the status of the state database. A legend follows the replica status.

10

Examples

The following two examples show how to set up the initial state database and create replicas. The first example describes the procedure for a new system, and the second describes the procedure for an existing system.

State Database Replicas on a New System

This example shows how to set up an initial state database and several replicas on a new system that has been delivered with four disks for user's data.

This example assumes that DiskSuite is installed. It is also assumed that `/dev/dsk/c0t0d0s7` and `/dev/dsk/c0t1d0s2` will contain all the user data. The other two disks, `/dev/dsk/c1t0d0s7` and `/dev/dsk/c1t1d0s2`, will mirror the disks that contain the user data.

To set up the initial state database follow these steps:

1. **Edit the `/etc/opt/SUNWmd/md.tab` file and add the following line:**

```
mddb01 /dev/dsk/c0t0d0s7 /dev/dsk/c0t1d0s2 /dev/dsk/c1t0d0s7 \
                /dev/dsk/c1t1d0s2
```

2. **Use the `metadb` command with the `-a` and `-f` options to activate the state database.**
   For example:

```
# /usr/opt/SUNWmd/sbin/metadb -a -f mddb01
```

3. **Concatenate the components into two submirrors by adding the following lines to the `/etc/opt/SUNWmd/md.tab` file:**

```
d10 2 1 /dev/dsk/c0t0d0s7 1 /dev/dsk/c0t1d0s2
d11 2 1 /dev/dsk/c1t0d0s7 1 /dev/dsk/c1t1d0s2
d12 -m d10
```

4. **Use the** `metainit` **command to initialize the concatenated submirrors and create the mirror.**

```
# /usr/opt/SUNWmd/sbin/metainit d10
# /usr/opt/SUNWmd/sbin/metainit d11
# /usr/opt/SUNWmd/sbin/metainit d12
# /usr/opt/SUNWmd/sbin/metattach d12 d11
```

State Databases on Existing Systems

This example shows how to add replicas on new disks that have been connected to a system that is currently running DiskSuite. In this example, the system already has been configured as shown in the previous example.

Two drives, `/dev/dsk/c0t2d0s2` and `/dev/dsk/c1t1d0s2`, are being added to the configuration. Two additional replicas are being added to the existing four replicas.

To add the drives and replicas, follow these steps:

1. **Edit the** `/etc/opt/SUNWmd/md.tab` **file and add the following line:**

```
mddb02 /dev/dsk/c0t2d0s2 /dev/dsk/c1t1d0s2
```

2. **Use the** `metadb` **command with only the** `-a` **option to activate the replicas.** For example:

```
# /usr/opt/SUNWmd/sbin/metadb -a mddb02
```

Or, to add the drives and replicas in one step:

♦ **Type the following on the command line:**

```
# /usr/opt/SUNWmd/sbin/metadb -a  \
  /dev/dsk/c0t2d0s2 /dev/dsk/c1t1d0s2
```

10

Expanding a File System 11

You can expand a mounted or unmounted UFS file system by using the utilities provided with DiskSuite. Only UFS file systems can be expanded using these utilities.

This chapter provides information about file system expansion with the DiskSuite software package. Use the following table to locate specific information.

File System Expansion Overview

Basically, expanding a file system occurs in two steps.

1. Disk space is added at the end of the metadevice using the DiskSuite dynamic concatenation facilities.
2. The file system is expanded using the `growfs` command.

Note – Once a file system is expanded, it cannot be shrunk.

Aborting a `growfs(1M)` command may cause a temporary loss of free space. The space can be recovered using the `fsck(1M)` command after the file system is unmounted using `umount(1M)`.

For metamirrors, each of the submirrors must be increased in size before you can expand the file system. If an error is reported while growing any of the submirrors, that error must be resolved before you can expand the file systems.

For metatrans devices, only the logging device or master device can be expanded; not the metatrans device itself. However, the `growfs(1M)` command should be run on the metatrans device.

Nonexpandable File Systems

A mounted file system cannot be expanded if any of the following conditions exist:

- When `acct(1M)` is activated and the accounting file is on the target file system.
- There is a local `swap` file in the target file system.
- C2 security is activated and the logging file is on the target file system.
- The file system is `root`, `/usr`, or `swap`.

Adding Components

When a component is available, the `metattach` command is used to add the component to the existing metadevice without interrupting service.

If more than one component is added in a single use of `metattach`, those components are striped. This stripe is then concatenated onto the metadevice.

Disk space can be expanded by one or all of the following:

- Adding a physical partition to a metadevice.
- Adding a stripe to a metadevice.
- Adding a physical partition or stripe to all submirrors of a metamirror.
- Adding partitions to a RAID device.

The `growfs` *Command*

The `growfs` command nondestructively expands a file system up to the size of the file system's physical device or metadevice. The `growfs` command is actually a "friendly" front-end to the `mkfs(1M)` command.

`growfs` write-locks (see `lockfs(1M)`) the file system when expanding a mounted file system. Access times are not kept while the file system is write-locked. The `lockfs` command can be used to check the file system lock status and unlock the file system in the unlikely event that `growfs` aborts without unlocking the file system.

The file system can be expanded to use only part of the additional disk space using the `growfs -s` option.

Examples

Several step-by-step examples that show how to expand a file system are provided in this section. These examples include:

- Expanding a nonmetadevice component
- Expanding a mounted file system
- Expanding a mounted file system to an existing metamirror
- Expanding an unmounted file system
- Expanding a mounted file system using stripes

Expanding a Nonmetadevice Component

The following example shows the expansion of a nonmetadevice component. In this example, the existing component is converted to a metadevice so additional components can be concatenated. This procedure necessitates unmounting the file system (`/var` is used as the example) and remounting the file system under the new metadevice.

1. **Edit the** `/etc/opt/SUNWmd/md.tab` **file, adding a line that defines the metadevice that will consist of the existing component (**`/dev/dsk/c1t0d0s3`**) and a new component (**`/dev/dsk/c2t0d0s3`**) to be concatenated onto the existing one.**

```
d8 2 1 /dev/dsk/c1t0d0s3 1 /dev/dsk/c2t0d0s3
```

2. **The file system on** `/var` **must now be unmounted, using the following command:**

```
# /sbin/umount /var
```

3. `metainit` **is now used to create the new metadevice.**

```
# /usr/opt/SUNWmd/sbin/metainit d8
```

4. **Edit the** `/etc/vfstab` **file to change the entry for the** `/var` **file system to be the newly defined metadevice name rather than the component name.** For instance, change the following line:

```
/dev/dsk/c1t0d0s3 /dev/rdsk/c1t0d0s3  /var  ufs  4  yes  –
```

   to read:

```
/dev/md/dsk/d8 /dev/md/rdsk/d8  /var  ufs  4  yes  –
```

5. **The file system can now be remounted, using** `mount`**.**

```
# /sbin/mount /var
```

6. **Use the** `growfs` **command to expand the file system.**

```
# /usr/opt/SUNWmd/sbin/growfs -M /var /dev/md/rdsk/d8
```

All of `/dev/md/rdsk/d8` will be used to expand the file system. The `–M` option allows `growfs` to expand a mounted file system. The `–s` option can be used to limit the size. During this expansion `/var` is not available for write

access because of the write-lock. Write accesses are transparently suspended and are restarted when `growfs` unlocks the file system. Read accesses are not affected, though access times are not kept while the lock is in effect.

Expanding a Mounted File System

The following example shows how to expand a file system mounted on `/var` by concatenating a single disk drive to an existing metadevice. In this example, the metadevice is named `d8`. The metadevice is not a metamirror.

1. **Use the `metattach` command to dynamically concatenate the new drive to the end of the existing metadevice.**

```
# /usr/opt/SUNWmd/sbin/metattach d8 /dev/dsk/c0t1d0s2
```

2. **Use the `growfs` command to expand the file system.**

```
# /usr/opt/SUNWmd/sbin/growfs -M /var /dev/md/rdsk/d8
```

All of `/dev/md/rdsk/d8` will be used to expand the file system. The `–M` option allows `growfs` to expand a mounted file system. The `–s` option can be used to limit the size. During this expansion `/var` is not available for write access because of the write-lock. Write accesses are transparently suspended and are restarted when `growfs` unlocks the file system. Read accesses are not affected, though access times are not kept when the lock is in effect.

Expanding a Mounted File System to an Existing Metamirror

The following example shows how to expand a mounted file system by concatenating three disk drives to an existing three-way metamirror. In this example, the metamirror is named `d8` and contains three submirrors named `d9`, `d10`, and `d11`.

1. **Use the** `metattach` **command to dynamically concatenate the new drives to the end of each existing submirror within the metamirror.**
   `metattach` must be run for each of the submirrors.

```
# /usr/opt/SUNWmd/sbin/metattach d9 /dev/dsk/c0t2d0s5
# /usr/opt/SUNWmd/sbin/metattach d10 /dev/dsk/c0t3d0s5
# /usr/opt/SUNWmd/sbin/metattach d11 /dev/dsk/c0t4d0s5
```

   The metamirror will automatically grow when the last submirror is dynamically concatenated. The metamirror grows to the size of the smallest submirror.

2. **Use the** `growfs` **command to expand the file system.**

```
# /usr/opt/SUNWmd/sbin/growfs -M /var /dev/md/rdsk/d8
```

All of `/dev/md/rdsk/d8` will be used to expand the file system. The `-M` option allows `growfs` to expand a mounted file system. The `-s` option can be used to limit the size. During this expansion `/var` is not available for write access because of the write-lock. Write accesses are transparently suspended and are restarted when `growfs` unlocks the file system. Read accesses are not affected, though access times are not kept when the lock is in effect.

Expanding an Unmounted File System

The following example shows how to expand an unmounted file system. This example has a single disk drive, like the first example. Because the file system is not mounted, no file system locking occurs. The metadevices in this example must have already been initialized using `metainit`.

1. **Use the** `metattach` **command to dynamically concatenate the new drive to the end of the existing metadevice.**

```
# /usr/opt/SUNWmd/sbin/metattach d8 /dev/dsk/c0t1d0s5
```

2. **Use the** `growfs` **command to grow the file system.**

```
# /usr/opt/SUNWmd/sbin/growfs /dev/md/rdsk/d8
```

Expanding a Mounted File System Using Stripes

The following example shows how to expand a mounted file system by concatenating four disk drives to an existing metadevice. These four drives will be added to the metadevice as a stripe. In this example, the metadevice is named `d8`.

1. **Use the `metattach` command to dynamically concatenate the new IPI drive to the end of the existing metadevice.**

```
# /usr/opt/SUNWmd/sbin/metattach d8 /dev/dsk/c0t1d0s5 \
       /dev/dsk/c0t2d0s5 /dev/dsk/c0t3d0s5 /dev/dsk/c0t4d0s5
```

2. **Use the `growfs` command to exapnd the file system.**

```
# /usr/opt/SUNWmd/sbin/growfs -M /var /dev/md/rdsk/d8
```

All of `/dev/md/dsk/d8` will be used to expand the file system. The `–M` option allows `growfs` to expand a mounted file system. The `–s` option can be used to limit the size. During this expansion `/var` is not available for write access because of the write-lock. Write accesses are transparently suspended and are restarted when `growfs` unlocks the file system. Read accesses are not affected, though access times are not kept when the lock is in effect.

11

Configuration Guidelines 12

This chapter discusses configuration information for the Solstice DiskSuite product. Use the following table to locate specific information.

Performance Considerations

DiskSuite is not intended primarily to improve disk performance. It may do so in some cases, but it is also possible that a badly designed configuration will degrade performance. This section offers tips for getting good performance from DiskSuite.

Some performance considerations include the following:

- When concatenating, don't concatenate partitions on a single drive. For example, don't concatenate `/dev/dsk/c0t1d0s4` and `/dev/dsk/c0t1d0s7`; you can do so, but it will hurt performance. If you want to concatenate these devices, make sure they are far apart in the list of devices in the `md.tab` file. For example:

```
d8 4 1 /dev/dsk/c0t1d0s4 1 /dev/dsk/c0t2d0s0 \
     1 /dev/dsk/c0t1d0s7 1 /dev/dsk/c0t0d0s0
```

  will tend to have better performance than:

```
d8 4 1 /dev/dsk/c0t1d0s4 1 /dev/dsk/c0t1d0s7 \
     1 /dev/dsk/c0t2d0s0 1 /dev/dsk/c0t0d0s0
```

  due to behavior of concatenations.

- When mirroring, don't have multiple copies of the data on the same drive. Always try to use at least two different drives for mirroring, since writes to the same drive contend for the same resources, and the failure of the one drive would mean the loss of all data.
- When logging UFS file systems, try to have the logging and master devices on different drives.

  However, if one of the disks or controllers in a metamirror is heavily loaded by other activity (for example, it contains another file system on another of its partitions), you may want to use the `-r` option. This mode will force the metadisk driver to read only from one partition, thus avoiding further overloading the other partition. (Note that the metadisk driver does not automatically choose the least loaded partition. You must specify this by making the less heavily-loaded metadevice the first submirror.)
- Try to have all drives in a metadevice on separate data paths. For SCSI drives, this means separate host adaptors. For IPI drives, this means separate controllers. By having the metadevice spread the I/O load over several controllers, performance and availability will be improved.
- Experiment with the read options of the metadisk driver in a metamirror. The default mode is to alternate reads in a round-robin fashion among the disks. This is the default because it tends to work best for UFS multi-user, multi-process activity.

- In some circumstances, using the `-g` option will improve performance by minimizing head motion and access times. This option is most effective when there is only one partition per disk, when only one process at a time is using the partition/file system, and when I/O patterns are highly sequential or when all accesses are reads.
- Don't mix disks or controllers of widely varying performance or technologies in a single metadevice or metamirror. Particularly in old SCSI or SMD storage devices, different models or brands of disk or controller can have widely varying performance. Mixing the different performance levels in a single metadevice or metamirror can cause performance to degrade significantly.

Availability Considerations

The metadisk driver's mirroring capability provides insurance against data loss with single point failures in a UNIX system. This section provides a few tips to ensure that you'll actually realize that safety.

Some considerations to be followed to ensure availability of data are:

- The `/etc/opt/SUNwmd/mddb.cf` or `/etc/opt/SUNWmd/md.cf` files should never be edited or removed.
- Make sure that the `/etc/opt/SUNWmd/md.cf` file is backed up frequently.
- When concatenating or striping components, mirror them if you have the spare disk capacity. Concatenating partitions reduces effective reliability somewhat, so mirroring any critical data stored in a concatenated metadevice is a good idea.
- When logging UFS file systems, mirror the logging device.
- When mirroring, keep the components of different submirrors on separate disks. Data protection is diminished considerably if components of two or more submirrors of the same mirrored metadevice are on the same disk.
- When mirroring, define the metadevices with components on separate controllers, if possible. Controllers and associated cables may fail more often than disks, so organize the submirrors of your metamirrors so that the submirrors are as independent as possible. This also helps performance.

- When creating RAID devices, define with components on separate controllers, if possible. Controllers and associated cables may fail more often than disks, so organize the submirrors of your metamirrors so that the submirrors are as independent as possible. This also helps performance.
- If the metadisk driver takes a metadevice offline, consider unmounting other file systems mounted on the disk where the failure occurred. Since each disk partition is independent, up to eight file systems may be mounted on a single disk. If the metadisk driver has “trapped” a failure (as indicated by `metastat`), other partitions on the same disk will likely experience failures soon. File systems mounted directly on disk partitions do not have the protection of metadisk driver error handling, and leaving such file systems mounted can leave you vulnerable to crashing the system and losing data.
- Minimize the amount of time you run with submirrors disabled or offline. During resyncing and online backup intervals, you do not have the full protection of mirroring. Your system administrator should run `metastat` regularly on any servers using the metadisk driver to check for metadevices that are not functioning properly.
- Do not mount file systems on an underlying component of a metadevice. If a physical partition is used for a metadevice of any kind, you must not mount that partition as a file system. If possible, unmount any physical device you intend to use as a metadevice before you activate it.
- After a component is defined as a metadevice and activated, do not use it for another purpose.
- Use the block device name (for example, `/dev/md/dsk/d8`) for block device functions such as `mount(1M)` and `umount(1M)`; use the raw device name (`/dev/md/rdsk/d8`) for raw device functions such as `newfs(1M)`, `dd(1M)`, or `fsck(1M)`.
- Stripes and concatenations cannot be defined using a metadevice as a component; only physical partitions can be used as components of a stripe or concatenation.
- For mirrored metadevices, only the metamirror can be mounted. An error message is returned if you try to mount a submirror directly, unless the submirror is offline and mounted read-only. However, you will not get an error message if you try to mount a physical partition that is a component of a metadevice; this could destroy data and crash the system.

- Two DiskSuite startup files are installed in system startup directories to effect automatic rebooting and resyncing of metadevices: `/etc/rcS.d/S35SUNWmd.init` and `/etc/rc2.d/S95SUNWmd.sync`. See Chapter 2, “Installation and Setup,” for detailed instructions on setting up DiskSuite.
- If you want all `swap` space to be mirrored, you should use `swap -l` to check for `swap` devices. Partitions specified as `swap` must be mirrored separately.

Capacity Considerations

DiskSuite increases system capacity by supporting the concatenation, striping, or RAIDing of components into metadevices. These metadevices can then be used as either file systems or raw devices.

Some considerations that deal with capacity are:

- Components of differing physical size or geometry can be used effectively with the metadisk driver; what is important in defining metamirrors is that the mirrored metadevices be the same size, or nearly so. If you define a metamirror with two different sized metadevices, the metamirror will be the size of the smaller of the two metadevices; the extra space on the larger metadevice is unused.

  If you define a metamirror with one metadevice, the metamirror will be the size of that metadevice, regardless of the size of metadevices attached later. You will not be allowed to attach a smaller metadevice.
- When metadevice state database replicas are placed on a component used by a metadevice, the capacity of that metadevice is reduced. The space occupied by a replica is rounded up to the next cylinder boundary and this space is skipped by the metadevice.
- On components with labels (those starting at cylinder 0), the first cylinder of a component is skipped when the component is not the first component of a metadevice.
- All components of a stripe are the size of the smallest component. The components are rounded down to the nearest multiple of interlace size. So, if different size components are used within in a stripe, then disk capacity will be limited to a multiple of the smallest.
- When you’re using the file system logging facility, remember that larger logs result in greater concurrency.

12

Labeled Partitions

All physical devices must have one disk label. The label is normally created by the `install` software, the `format` program, or the `fmthard` program. The label can appear on more than one of the logical partitions that are defined in the label.

Physical partitions that contain a label should not allow a user to write to the block that contains the label. Normally this is block 0. UNIX device drivers allow a user to overwrite this label.

Security Considerations

DiskSuite does not provide an audit trail for any reconfiguration of metadevices that may be performed on the system. This means that DiskSuite does not support C2 security.

Compatibility Considerations

DiskSuite is compatible with most other Sun Microsystem products. However, there are some limitations and considerations.

Some considerations that deal with compatibility are:

- Machines using Solstice DiskSuite 4.0 must be running the Solaris 2.1 release or a later Solaris 2.*x* release.
- To use the DiskSuite UFS file logging or diskset features, you must be running the Solaris 2.4 or a later Solaris 2.*x* release.
- When using Sun Prestoserve™ on a system running DiskSuite, you should not use Prestoserve features on metamirrors or their submirrors; or on metatrans devices if the logging or master devices are mirrored.
- DiskSuite is compatible with the Online: Backup 2.0 unbundled product.
- The DiskSuite diskset feature is not supported on x86 systems.

Solstice DiskSuite Files A

Introduction

There are four system files associated with DiskSuite. These files are used by the various programs. This appendix gives a description of each file, offers instructions for creating these files, and provides some basic examples.

Three system files associated with DiskSuite are located in `/etc/opt/SUNWmd`. These include:

- `mddb.cf`
- `md.tab`
- `md.cf`

The fourth system file associated with DiskSuite is located in `/kernel/drv`:

- `md.conf`

The `mddb.cf` *File*

The `mddb.cf` file keeps track of metadevice state database replica locations. Each metadevice state database has a unique entry in this file. This file is automatically generated. No action on the part of the user is required.

Warning – The `mddb.cf` file should never be edited or removed.

When the layout of the state database locations change, two actions occur:

- An entry is made in the `mddb.cf` file that tells the locations of all the state databases.
- The identical information is edited into the `/etc/system` file.

In the following example, there are two state database replicas located on each of the devices entered in the `mddb.cf` file. On `/dev/dsk/c0t0d0s0`, state database replicas start on block 16 and on block 1050. The default size for state database is 1034 blocks.

An example of a `mddb.cf` file follows:

```
#metadevice database location file do not hand edit
#driver minor_t daddr_t checksum
 sd     27      16      -216
 sd     27      1050    -1250
 sd     83      16      -272
 sd     83      1050    -1306
 sd     19      16      -208
 sd     19      1050    -1242
```

The `driver` field indicates the device driver for the disk. The `minor_t` field represents the device and the partition on which the state database replica resides. The `daddr_t` field is the first block used by the state database replica. The `checksum` field is used to insure the other fields are valid.

The `md.tab` *File*

The `md.tab` file is the input file used by `metainit`, `metadb`, and `metahs`. Each metadevice and each hot spare pool may have unique entries in this file.

The standard metadevice entry name begins with `d` and is followed by a number. By default, there are 128 unique metadevices in the range 0 to 127.

A metadevice may be a concatenation or a stripe of component partitions.

Metamirrors can also be defined in the `md.tab` file. A metamirror is a special type of metadevice that is made up of one or more other metadevices. Each metadevice within a metamirror is a called a submirror. Metamirrors have names of the same form as other metadevices.

Hot spare pools may also be defined in the `md.tab` file. Hot spares can be added as part of a hot spare pool. If a component is designated as a hot spare it can not be used in a submirror or another metadevice.

Once metadevices are specified in the `md.tab` file, they can be activated using the `metainit` command.

The `md.tab` file might not contain an entry for all initialized metadevices, since metadevices can be initialized using `metainit` command line options.

An example `md.tab` file is shown below. The example shows numerous metadevices, metamirrors, hot spares, and hot spare pools.

As shown in the example, the use of tabs, spaces, comments (using the pound sign character) and continuation of lines (using the backslash character) are accepted.

```
# DiskSuite configuration file

# mirror of root
d1 -m d5 2
d5 1 1 /dev/dsk/c0t3d0s0
d6 1 1 /dev/dsk/c1t3d0s0

# mirror of swap
d0 -m d7
d7 1 1 /dev/dsk/c0t3d0s1
d8 1 1 /dev/dsk/c1t3d0s1

# mirror of /usr
# geometric reads are selected
d9 -m d11 -g
d11 1 1 /dev/dsk/c0t3d0s6 -h hsp001
d12 1 1 /dev/dsk/c1t3d0s6 -h hsp002

# define hot spare pools
hsp001 /dev/dsk/c1t1d0s2
hsp002 /dev/dsk/c0t1d0s2
```

For additional information on setting up a `md.tab` file, refer to the Chapter 4, “Concatenating and Striping,” Chapter 5, “Mirroring,” Chapter 6, “UFS Logging,” Chapter 7, “Hot Spares,” and the `md.tab(4)` manual page.

A

The `md.cf` *File*

The `md.cf` file is automatically updated whenever the configuration is changed by the user. This is basically a backup file intended for disaster recovery only.

Warning – The `md.cf` file should never be manually edited.

The output from this file is similar to that displayed when you run `metastat -p`.

The `md.conf` *File*

The `md.conf` file is used by the metadisk driver when it is initially loaded. The only modifiable field in this file is "`nmd`," which represents the number of metadevices supported by the driver. An example of the default entry follows:

```
name="md" parent="pseudo" nmd=128 md_nsets=4;
```

If you modify the `nmd` field, you must perform a reconfiguration boot (`boot -r`) to build the metadevice names.

Warning – If you lower this number, any metadevice existing between the old number and the new number may not be persistent.

The default `nmd` value is 128. Other values are supported up to 1024.

If you create a large number of metadevices, the state database replicas may eventually be too small. If this happens, try adding larger replicas (with the `-l` option and `metadb` command) and then removing the smaller replicas. For example, if you create 1024 metadevices, database replicas should be increased with to 2500 sectors.

Note – When you add larger numbers of metadevices, you may begin to see some performance degradation while administering metadevices.

If you are increasing the number of metadevices to gain a larger namespace for partitioning the types of devices within certain numeric ranges, but you are creating fewer than 128 metadevices, you should not see any performance degradation. In this case, you should not have to add larger replicas.

A

Solstice DiskSuite Messages B

Introduction

This appendix contains the error and log messages displayed by the metadevice utilities of DiskSuite 4.0.

Errors that deal with command usage and other simple error messages are not documented in this appendix. All DiskSuite error messages are displayed in the following format:

program name: *host*: [*optional1*:] *name*: [*optional2*]: *error message...*

where:

- *program name*: is the name of the application name and version being used (for example, *DiskSuite 4.0*).
- *host*: is the host name of the machine on which the error occured (for example, *blue*).
- [*optional1*]: is an optional field containing contextual information for the specific error displayed (ie. mountpoint or which daemon returned the error).
- *name*: is the command name which generated the error message (for example, `metainit`).
- [*optional2*]: is a second optional field containing additional contextual information for the specific error displayed
- *error message...* is the error message itself (as listed in this appendix).

B

For the purpose of this appendix, only the final portion (*error message...*) of each error message is listed.

The log messages listed near the back of this appendix are divided into three categories:

- Notice log messages
- Warning log messages
- Panic log messages

Error Messages

The error messages displayed by DiskSuite are listed in alphabetical order below. The message is preceded by some or all of the variables described in the previous section. Other variables included in these messages indicate the following:

- *nodename* is the name of a specific host.
- *drivename* is the name of a specific drive.
- *metadevice* is the number of a specific metadevice device or hot spare pool.
- *setname* is the name of a specific diskset.
- *num* is a number.

```
add or replace failed, hot spare is already in use
```

The hot spare that is being added or replaced is already in the hot spare pool.

```
administrator host nodename can't be deleted, other hosts still in
set. Use -f to override
```

The host which owns the diskset cannot be deleted from the diskset without using the `-f` option to override this restriction. When the -f option is used, all knowledge of the diskset is removed from the local host. Other hosts within the diskset are unaware of this change.

```
administrator host nodename deletion disallowed in one host admin
mode
```

The administrator host is the host which has executed the command. This host cannot be deleted from the diskset if one or more host in the diskset are unreachable.

```
already has log
```

The specified metatrans device already has an attached logging device.

```
already used in metadevice
```

The specified component is currently being used in the *metadevice*.

```
attempt to detach last running submirror
```

An attempt was made to detach the last submirror. The operation would result in an unusable mirror. DiskSuite does not allow a `metadetach` to be performed on the last submirror.

```
attempt an operation on a submirror that has erred components
```

An attempt was made to take a submirror offline or detach a submirror that contains the data. The other submirrors have erred components. If this operation were allowed, the mirror would be unusable.

```
attempt an operation on a submirror in illegal state
```

An attempt was made to take a submirror offline that is not in the OKAY state or to online a submirror that is not in the offlined state. Use the `-f` option if you really need to offline a submirror that is in a state other than OKAY.

```
attempt to replace a component on the last running submirror
```

An attempt was made to replace a component in a one-way mirror.

```
attempted to clear mirror with submirror(s) in invalid state
```

The user attempted to use the `metaclear` command on a metamirror that contained submirrors that weren't in the OKAY state (Needs maintenance state). If the metamirror must be cleared, the submirrors must also be cleared. Use `-r` (recursive) to clear all the submirrors, or use `-f` (force) to clear a metamirror containing submirrors in the Needs maintenance state.

```
can't attach labeled submirror to unlabeled mirror
```

An attempt was made to attach a labeled submirror to an unlabeled mirror. A labeled metadevice is a device whose first component starts at cylinder 0. To prevent the submirror's label from being corrupted, DiskSuite does not allow labeled submirrors to be attached to unlabeled mirrors.

```
can't find component in unit
```

An attempt was made to replace or enable a component that did not exist in the specified metadevice.

```
can’t find submirror in mirror
```

An attempt was made to either `metaonline(1M)`, `metaoffline(1M)`, or `metadetach(1M)` the submirror, d*num*. The submirror is not currently attached to the specified metamirror causing the command to fail.

```
can’t include device dev, it already exists in dnum
```

An attempt was made to use the device *dev* in a new metadevice and it already is used in the metadevice d*num*.

```
can’t include device dev, it overlaps with a device in dnum
```

The user has attempted to use device *dev* in a new metadevice which overlaps an underlying device in the metadevice, d*num*.

```
cannot delete the last database replica in the diskset
```

An attempt was made to delete the last database replica in a diskset. To remove all database replicas from a diskset, delete all drives from the diskset.

```
cannot enable hotspared device
```

An attempt was made to perform a `metareplace -e` (enable) on an underlying device which is currently hot spared. Try enabling the hot spare component instead.

```
can’t modify hot spare pool, hot spare in use
```

An attempt was made to modify the associated hot spare pool of a submirror, but the submirror is currently using a hot spare contained within the pool.

```
checksum error in mddb.cf file
```

The `/etc/opt/SUNWmd/mddb.cf` file has probably been corrupted or user-edited. The checksum this file contains is currently invalid. To remedy this situation: 1. Delete the `mddb.cf` file. 2. Delete a database replica. 3. Add back the database replica.

```
component in invalid state to replace \
     - replace “Maintenance” components first
```

An attempt was made to replace a component that contains the only copy of the data. The other submirrors have erred components. If this operation were allowed, the mirror would be unusable.

```
data not returned correctly from disk
```

After a replica of the state database is first created, it is read to make sure it was created correctly. If the data read does not equal the data written this message is returned. This results from unreported device errors.

```
device not in set
```

An attempt was made to use a component for a shared metadevice or shared hot spare pool whose drive is not contained within the diskset.

```
device in shared set
```

An attempt was made to use a component for a local metadevice or local hot spare pool whose drive is contained within the diskset. The drives in the local diskset are all those which are not in any shared disksets.

```
device is too small
```

A component (*dev*) in stripe *num* is smaller than the interlace size specified with the `-i` flag in the `md.tab` file.

```
device size num is too small for metadevice database replica
```

An attempt was made to put a database replica on a partition that is not large enough to contain it.

```
devices were not RAIDed previously or are specified in the wrong
order
```

An attempt was made to metainit a RAID device using the `-k` option. Either some of the devices were not a part of this RAID device, or the devices were specified in a different order than they were originally specified.

`drive` *drivename* `is in set` *setname*

An attempt was made to add the drive *drivename* to a diskset which is already contained in the diskset *setname*.

`drive` *drivename* `is in use`

An attempt was made to add the drive *drivename* to a diskset, however a slice on the drive is in use.

`drive` *drivename* `is not common with host` *nodename*

An attempt was made to add the drive *drivename* to a diskset, however, the device name or device number is not identical on the local host and the specified *nodename*; or the drive is not physically connected to both hosts.

`drive` *drivename* `is not in set`

An attempt was made to delete the drive *drivename* from a diskset and the diskset does contain the specified drive.

`drive` *drivename* `is specified more than once`

The same drive (*drivename*) was specifed more than once in the command line.

```
driver version mismatch
```

The utilities and the drivers are from different versions of the DiskSuite package. It is possible that either the last `SUNWmd` package added did not get fully installed (try running `pkgchk(1M)`), or the system on which DiskSuite was recently installed has not been rebooted since the installation.

```
failed to take ownership of a majority of the drives
```

Reservation of a majority of the drives was unsuccessful. It is possible that more than one host was concurrently attempting to take ownership of the same diskset. One host will succeed, and the other will receive this message.

```
growing of metadevice delayed
```

The attempted growth of a submirror has been delayed until a mirror resync finishes. The metamirror will be grown automatically upon completion of the resync operation.

```
has a metadevice database replica
```

An attempt was made to use a component (ie. for a hot spare) which contains a database replica.

```
host nodename already has a set numbered setnumber
```

An attempt was made to add a host *nodename* to a diskset which has a conflicting *setnumber*. Either create a new diskset with both hosts in the diskset, or delete one of the conflicting disksets.

```
host nodename already has set
```

An attempt was made to add a host *nodename* to a diskset which has a different diskset using the same name. Delete one of the disksets and recreate the diskset using a different name.

```
host nodename does not have set
```

An attempt was made to delete a host or drive from a set, but the host *nodename* has an inconsistent view of the diskset. This host should probably be forcibly (`-f`) deleted.

```
host nodename is already in the set
```

An attempt was made to add a host *nodename* which already exists within the diskset.

```
host nodename is modifying set - try later or restart rpc.metad
```

Either an attempt was made to perform an operation on a diskset at the same time as someone else, or a previous operation dropped core and the `rpc.metad` daemon should be restarted on host *nodename*.

```
host nodename is not in the set
```

An attempt was made to delete the host *nodename* from a diskset which does not contain the host.

```
host nodename is specified more than once
```

The same host (*nodename*) was specifed more than once in the command line.

```
host name nodename is too long
```

The name used for the host nodename is longer than DiskSuite accepts.

```
hotspare doesn't exist
```

An attempt was made to perform an operation on the hot spare *dev* and the specified hot spare does not exist.

```
hotspare in use
```

An attempt was made to perform an operation on the hot spare *dev* and the specified hot spare is in use.

```
hotspare isn't broken, can't enable
```

An attempt was made to enable a hot spare that is not in the broken state.

```
hotspare database create failure
```

An attempt to create a hot spare record in the metadevice state database failed. Run `metadb -i` to determine the cause of the failure.

```
hotspare pool database create failure
```

An attempt to create a hot spare pool record in the metadevice state database failed. Run `metadb -i` to determine the cause of the failure.

```
hotspare pool is busy
```

An attempt was made to delete the hot spare pool `hsp`*nnn* before removing all the hot spares associated with the specified hot spare pool.

```
hotspare pool is referenced
```

An attempt was made to delete the hot spare pool, `hsp`*nnn*, that is associated with a metadevice.

```
hotspare pool in use
```

An attempt was made to `metaclear(1M)` a hotspare pool without first removing its association with metadevices.

```
hotspare pool is already setup
```

An attempt was made to create a hot spare pool which already exists.

```
illegal option
```

An attempt was made to use an option which is not valid in the context of the specified metadevice or command.

```
in Last Erred state, errored components must be replaced
```

An attempt was made to replace or enable a component of a mirror in the “Last Erred” state when other components are in the “Erred” state. You must first replace or enable all of the components in the “Erred” state.

```
invalid RAID configuration
```

An invalid RAID device configuration entry was supplied to `metainit`, either from the command line or via the `md.tab` file.

```
invalid argument
```

An attempt was made to use an argument which is not valid in the context of the specified metadevice or command.

```
invalid column count
```

An invalid RAID configuration entry was supplied to `metainit`, either from the command line or via the `md.tab` file. Specifically, an invalid argument was provided with the `-o` option.

```
invalid interlace
```

An unsupported interlace value follows the `–i` option on a metadevice configuration line. The `–i` specifies the interlace size. The interlace size is a number (8, 16, 32) followed by either `k` for kilobytes, `m` for megabytes, or `b` for blocks. The units can be either uppercase or lowercase. This message will also appear if the interlace size specified is greater than 100 Mbytes.

```
invalid mirror configuration
```

An invalid mirror configuration entry was supplied to `metainit`, either from the command line or via the `md.tab` file.

```
invalid pass number
```

An attempt was made to use a pass number for a mirror that is within the 0 - 9 range.

```
invalid stripe configuration
```

An invalid stripe configuration entry was supplied to `metainit`, either from the command line or via the `md.tab` file.

```
invalid trans configuration
```

An invalid trans configuration entry was supplied to `metainit`, either from the command line or via the `md.tab` file.

```
invalid write option
```

An attempt was made to change the write option on a mirror using an invalid option. The legal strings are “serial” and “parallel.”

```
invalid hotspare pool
```

The metadevice configuration entry in the `md.tab` file has a `-h hsp`*nnn* and a `metainit` has not been performed on the hot spare pool.

```
invalid read option
```

The user has specified both the `-r` and `-g` options on the same metamirror.

```
invalid unit
```

The metadevice (submirror) passed to `metattach` is already a submirror. The metadevice may already be a submirror for another metamirror.

```
is a metadevice
```

The device *dev* being used is a metadevice and it should be a physical component.

```
is mounted on
```

The device *dev* in the metadevice configuration has a file system mounted on it.

hostname is not a nodename, but a network name

An attempt was made to add a host to a diskset without using the nodename found in the `/etc/nodename` file.

`is swapped on`

The device in the metadevice configuration is currently being used as a `swap` device.

`maximum number of nodenames exceeded`

An attempt was made to add more nodenames than DiskSuite allows in a diskset.

`maxtransfer is too small`

An attempt was made to add a component to a RAID device whose maxtransfer is smaller than the other components in the RAID device.

`metadevice in use`

An attempt was made to `metaclear(1M)` a submirror without first running `metaclear` on the metamirror in which it is contained.

```
metadevice is open
```

The metadevice (submirror) passed to `metattach` is already open (in-use) as a metadevice.

```
num1 metadevice database replicas is too many;the maximum is num2
```

An attempt was made to add more databases (*num*1) than the maximum allowed (*num*2).

```
metadevice database has too few replicas, can't create new
records
```

An attempt to create a metadevice record in the metadevice state database failed. Run `metadb -a` to add more database replicas.

```
metadevice database is full, can't create new records
```

An attempt to create a metadevice record in the metadevice state database failed. Run `metadb -a` (and `-s`) to add larger database replicas. Then delete the smaller replicas.

```
metadevice database replica exists on device
```

An attempt was made to use a component (ie. for a hot spare) which contains a database replica.

```
mirror has maximum number of submirrors
```

An attempt was made to attach more than the supported number of submirrors. The maximum supported number of submirrors is three.

```
must be owner of the set for this command
```

An attempt was made to perform an operation on a diskset or a shared metadevice on a host which is not the owner of the diskset.

```
must have at least 2 databases (-f overrides)
```

An attempt was made to delete database replicas, reducing the number of database replicas to a number less than two. To override this restriction, use the `-f` option.

```
must replace errored component first
```

An attempt was made to replace or enable a component of a mirror in the “Last Erred” state when other components are in the “Erred” state. You must first replace or enable all of the components in the “Erred” state.

```
no available set numbers
```

An attempt was made to create more disksets than DiskSuite allows.

```
no hotspare pools found
```

An `metahs` operation was attempted using the "`all`" argument when no hot spare pools meet the criteria for the operation.

```
no metadevice database replica on device
```

An attempt was made to delete non-existent database replicas.

```
no such set
```

An attempt was made to perform an operation on a diskset or a shared metadevice using a non-existent set name.

```
nodename of host nodename creating the set must be included
```

An attempt was made to create a diskset on the local host without adding the name of the local host to the diskset.

```
not a disk device
```

The component name specified is not a disk device name. For example, a CD-ROM device doesn't have the characteristics of a disk device.

```
not enough components specified
```

An invalid stripe configuration entry was supplied to `metainit`, either from the command line or via the `md.tab` file.

```
not enough stripes specified
```

Invalid stripe configuration entry was supplied to `metainit`, either from the command line or via the `md.tab` file.

```
not enough submirrors specified
```

Invalid mirror configuration entry was supplied to `metainit`, either from the command line or via the `md.tab` file.

```
not in local set
```

An attempt was made to create a local metadevice or local hot spare pool with a component whose drive is contained in a shared diskset.

```
not a metadevice
```

The specified device is not a metadevice. DiskSuite expected a metadevice name.

```
only slice 7 is usable for a diskset database replica
```

An attempt was made to add a database replica for a shared diskset on a component other than Slice 7.

```
only the current owner nodename may operate on this set
```

An attempt was made to perform an operation on a diskset or a shared metadevice on a host which is not the owner of the diskset.

```
only valid action is metaclear
```

The initialization of a RAID device has failed. Use the `metaclear` command to clear the RAID device.

```
operation would result in no readable submirrors
```

An operation was attempted on a component or submirror that contains the only copy of the data. The other submirrors have erred components. If this operation were allowed, the mirror would be unusable.

```
operation requires -f (force) flag
```

Due to the components within the RAID device being in the “Maintenance” or “Last Erred” state, the force flag (`-f`) is required to complete the operation.

`overlaps with device in` *metadevice*

Overlapping slices are not allowed in metadevices or hot spare pools.

`replace failure, new component is too small`

An attempt to use `metareplace` failed because the new component is too small to replace the old component.

`reserved by another host`

An attempt was made to add a currently reserved drive to a diskset.

`resync in progress`

The mirror operation failed because a resync is being performed on the specified metamirror. Retry this operation when the resync is finished.

`set` *setname* `is out of date - cleaning up - take failed`

The diskset *setname* is out of data with respect to the other host's view. This error should occur only after one-host administration.

```
set lock failed - bad key
```

Either an operation on a diskset is currently being performed by another host, or an operation on a diskset was aborted. In the latter case, `rpc.metad` should be killed.

```
set name contains invalid characters
```

An attempt was made to use illegal characters to name a diskset.

```
set name is in-use or invalid on host nodename
```

The diskset name selected is already in use on host *nodename* or contains characters not considered valid in a diskset name.

```
set name is too long
```

An attempt was made to create a diskset using more characters in the diskset name than DiskSuite will accept.

```
set unlock failed - bad key
```

The diskset is locked and the user does not have the key. This may require killing `rpc.metad`.

```
side information missing for host nodename
```

The diskset is incomplete. Kill `rpc.metad` on all hosts and then retry the operation.

```
slice 7 is not usable as a metadevice component
```

An attempt was made to use Slice 7 in a shared metadevice or shared hot spare pool. Slice 7 is reserved for database replicas only.

```
submirror too small to attach
```

The metadevice passed to `metattach` is smaller than the metamirror to which it is being attached.

```
stale databases
```

The user attempted to modify the configuration of a metadevice when at least half the metadevice state database replicas were not accessible.

```
syntax error
```

An invalid metadevice confguration entry was provided to `metainit` from the command line or via the `md.tab` file.

```
there are no existing databases
```

To create any metadevices or hot spare pools, database replicas must exist. See `metadb(1M)` for information on the creation of database replicas.

```
unable to delete set, it still has drives
```

An attempt was made to delete the last remaining host from a diskset while drives still exist in the diskset.

```
unit already set up
```

The user requested that a metadevice d*num* be initialized when d*num* is already set up.

```
unit is not a concat/stripe
```

An attempt was made to perform a concat/stripe specific operation on a metadevice that is not a concat/stripe.

```
unit is not a mirror
```

An attempt was made to perform a mirror specific operation on a metadevice that is not a mirror.

```
unit is not a RAID
```

An attempt was made to perform a RAID specific operation on a metadevice that is not a RAID device.

```
unit is not a trans
```

An attempt was made to perform a metatrans specific operation on a metadevice that is not a metatrans device.

```
unit not found
```

An attempt was made to perform an operation on a non-existent metadevice.

```
unit not set up
```

An attempt was made to perform an operation on a non-existent metadevice.

```
waiting on /tmp/.mdlock
```

Some other metadevice utility is currently in progress and the lock cannot be accessed at this time. DiskSuite utilities are serialized using the `/tmp/.mdlock` file as a lock. If you determine that there are no other utilities currently running, you may want to remove this lock file.

B

Log Messages

The log messages displayed by DiskSuite are listed in alphabetical order below. Each message is always preceded with "`md:`" The variables in these messages indicate the following:

- *dev* is a device name.
- d*num* is a metadevice name.
- *num* is a number.
- *state* is a metatrans device state
- *trans* is either "logging" or "master"

Note – When the initial portion of a message begins with a variable, the message is alphabetized by the first word following the variable.

Notice Log Messages

```
Could not load misc /dev
```

The named misc module is not loadable. It is possibly missing, or something else has been copied over it.

```
db: Parsing error on 'dev'
```

The set command in `/etc/system` for the `mddb.bootlist`<*number*> is not in the correct format. Run `metadb -p` to place the correct set commands into the `/etc/system` file.

```
dnum: Hotspared device dev with dev
```

The first device name listed has been hot spare replaced with the second device name listed.

```
dnum: Hotspared device dev(num, num) with dev(num, num)
```

The first device number listed has been hot spare replaced with the second device number listed.

```
dnum: no mem for property dev
```

Memory could not be allocated in the `prop_op` entry point.

Warning Log Messages

```
dnum: Cannot load dev driver
```

The underlying named driver module is not loadable (for example, `sd`, `id`, `xy`, or a third-party driver). This could indicate that the driver module has been removed.

```
Open error of hotspare dev
Open error of hotspare dev(num, num)
```

The named hot spare is not openable, or the underlying driver is not loadable.

```
dnum: read error on dev
dnum: write error on dev
```

A read or write error has occurred on the specified metadevice at the specified device name. This happens if any read or write errors occur on a metadevice.

```
dnum: read error on dev(num, num)
dnum: write error on dev(num, num)
```

A read or write error has occurred on the specified metadevice at the specified device number. This happens if any read or write errors occur on a metadevice.

```
dnum: read error on dnum
dnum: write error on dnum
```

A read or write error has occurred on the specified metadevice at the specified device number. This happens if any read or write errors occur on a metadevice.

```
State database commit failed
State database delete failed
```

These messages occur when there have been device errors on components where the state database replicas reside. These errors only occur when more than half of the replicas have had errors returned to them. For example, if you have three components with state database replicas and two of the components report errors, than these errors may occur. The state database commit or delete is retried periodically. If the replica is added, the commit or delete will finish and the system will be operational. Otherwise, the system will time out and panic.

```
State database is stale
```

This message occurs when there are not enough usable replicas for the state database to be able to update records in the database. All accesses to the metadevice driver will fail. To fix this problem, add more replicas or delete unaccessible replicas.

```
trans device: read error on dnum
trans device: write error on dnum
```

A read or write error has occurred on the specified logging or master device at the specified metadevice. This happens if any read or write errors occur on a logging or master device.

```
trans device: read error on dev
trans device: write error on dev
```

A read or write error has occurred on the specified logging or master device at the specified device name. This happens if any read or write errors occur on a logging or master device.

```
trans device: read error on dev(num, num)
trans device: write error on dev(num, num)
```

A read or write error has occurred on the specified logging or master device at the specified device number. This happens if any read or write errors occur on a logging or master device.

```
logging device: dnum changed state to state
logging device: dev changed state to state
logging device: dev(num, num) changed state to state
```

The logging device and its associated master device(s) have changed to the specified state(s).

Panic Log Messages

```
State database problem
```

A failed metadevice state database commit or deletion has been retried the default 100 times.

```
dnum: Unknown close type
dnum: Unknown open type
```

A metadevice is being opened/closed with an unknown open type (OTYP).

B

Recovery From Failed Boots

Introduction

Because DiskSuite enables you to mirror `root`, `swap`, and `/usr`, special problems can arise when you are booting your system, either if a hardware failure occurs or through operator error. This appendix presents examples of such problems and provides possible solutions.

Improper `/etc/vfstab` *Entries*

A common problem that prevents the system from booting is failing to make proper entries in the `/etc/vfstab` file. While this problem may seem disastrous, the solution is actually fairly simple. The following example shows how you can edit the `/etc/vfstab` file to recover from a failed boot.

If you have failed to make the proper entry in the `/etc/vfstab` file when mirroring `root`, the machine will appear at first to be booting properly. In the following example, `root` is mirrored with a two-way mirror. The `root` entry in `/etc/vfstab` has somehow reverted back to the original component of the file system, but the information in `/etc/system` still shows booting to be off of a metadevice. The most likely reason for this to occur is that the `metaroot` command was not used to maintain `/etc/system` and `/etc/vfstab`, or an old copy of `/etc/vfstab` was copied back.

To remedy this situation, you need to edit `/etc/vfstab` while in single-user mode.

The incorrect `/etc/vfstab` file would look something like the following:

```
#device             device              mount    FS   fsck  mount  mount
#to mount           to fsck             point    type pass  at boot
options
#
/dev/dsk/c0t3d0s0 /dev/rdsk/c0t3d0s0 /       ufs  1    no      -
/dev/dsk/c0t3d0s1 -                 -       swap -    no      -
/dev/dsk/c0t3d0s6 /dev/rdsk/c0t3d0s6 /usr    ufs  2    no      -
#
/proc             -                 /proc   proc -    no      -
fd                -                 /dev/fd fd   -    no      -
swap              -                 /tmp    tmpfs-    yes     -
```

Because of the errors, you automatically go into single-user mode when the machine is booted:

```
ok boot
Booting from: sd(0,0,0)
SunOS Release 5.1 Version Generic [UNIX(R) System V Release 4.0]
Copyright (c) 1983-1992, Sun Microsystems, Inc.
...
Hostname: demo
dump on /dev/dsk/c0t3d0s1 size 34816K
mount: /dev/dsk/c0t3d0s0 is not this fstype.
setmnt: Cannot open /etc/mnttab for writing

INIT: Cannot create /var/adm/utmp or /var/adm/utmpx

INIT: failed write of utmpx entry:"  "

INIT: failed write of utmpx entry:"  "

INIT: SINGLE USER MODE

Type Ctrl-d to proceed with normal startup,
(or give root password for system maintenance):
Entering System Maintenance Mode

SunOS Release 5.1 Version Generic [UNIX(R) System V Release 4.0]
```

At this point, `root` and `/usr` are mounted read-only. Follow these steps:

1. **Run `fsck` and remount `root` read/write so you can edit the `/etc/vfstab` file.**

Note – Be careful to use the correct metadevice for root.

```
# fsck /dev/md/rdsk/d0
** /dev/md/rdsk/d0
** Currently Mounted on /
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
2274 files, 11815 used, 10302 free (158 frags, 1268 blocks, 0.7%
fragmentation)

# mount -o rw,remount /dev/md/dsk/d0 /
mount: warning: cannot lock temp file </etc/.mnt.lock>
```

2. **Edit the `/etc/vfstab` file to contain the correct metadevice entries.**

```
# vi /etc/vfstab
```

The `root` entry in the `/etc/vfstab` file should be edited to appear as follows:

```
#device             device               mount    FS   fsck  mount  mount
#to mount           to fsck                point    type  pass at boot
options
#
/dev/md/dsk/d0     /dev/md/rdsk/d0     /        ufs   1     no       –
/dev/dsk/c0t3d0s1  –                   –        swap  –     no       –
/dev/dsk/c0t3d0s6  /dev/rdsk/c0t3d0s6  /usr     ufs   2     no       –
#
/proc              –                   /proc    proc  –     no       –
fd                 –                   /dev/fd  fd    –     no       –
swap               –                   /tmp     tmpfs –     yes      –
```

3. Reboot the machine with the `reboot` command.

```
# reboot
```

Stale Metadevice Database

In this example, a disk which contains half of the database replicas and submirrors for `root`, `swap`, and `/usr` fails. Because half of the replicas are missing, the system cannot be rebooted.

Solving this problem involves::

1. Deleting the stale replicas and reboot
2. Repairing the disk
3. Adding back the database replicas
4. Re-enabling the broken submirrors

To remedy the example situation, you would perform the following steps:

1. Boot the machine to determine which replicas are down.

```
ok boot
Booting from: sd(0,0,0)
...
SunOS Release 5.1 Version Generic [UNIX(R) System V Release 4.0]
Copyright (c) 1983-1992, Sun Microsystems, Inc.
...
WARNING: md: State database is stale
...
metainit: stale databases
Insufficient metadevice database replicas
located. Use metadb to delete databases which
are no longer in existence. Exit the shell
when done to continue the boot process.

Type Ctrl-d to proceed with normal startup,
(or give root password for system maintenance):
Entering System Maintenance Mode

SunOS Release 5.1 Version Generic [UNIX(R) System V Release 4.0]
```

2. Use the `metadb` command to look at the database.

```
# /usr/opt/SUNWmd/sbin/metadb -i
   flags      first blk      block count
   a m  p  lu   16            1034             /dev/dsk/c0t3d0s3
   a   p  l     1050          1034             /dev/dsk/c0t3d0s3
    M  p       unknown     unknown             /dev/dsk/c1t2d0s3
    M  p       unknown     unknown             /dev/dsk/c1t2d0s3
 o - replica active prior to last mddb configuration change
 u - replica is up to date
 l - locator for this replica was read successfully
 c - replica's location was in /etc/opt/SUNWmd/mddb.cf
 p - replica's location was patched in kernel
 m - replica is master, this is replica selected as input
 W - replica has device write errors
 a - replica is active, commits are occurring to this replica
 M - replica had problem with master blocks
 D - replica had problem with data blocks
 F - replica had format problems
 S - replica is too small to hold current data base
 R - replica had device read errors
```

3. **Delete the stale database replicas using the `-d` option to the `metadb` command.**
   Since, at this point, the `root` file system is read-only, ignore the `mddb.cf` error messages:

```
# /usr/opt/SUNWmd/sbin/metadb -d /dev/dsk/c1t2d0s3
metadb: stale databases
metadb: /etc/opt/SUNWmd/mddb.cf.new: Read-only file system
metadb: databases installed but kernel not patched
        and new mddb.cf file not generated
metadb: could not open temp mddb.cf
Usage:  metadb [-s setname] -a [options] mddbnnn
        metadb [-s setname] -a [options] device ...
        metadb [-s setname] -d [options] mddbnnn
        metadb [-s setname] -d [options] device ...
        metadb [-s setname] -i
        metadb -p [options] [ mddb.cf-file ]
options:
-c count       number of replicas (for use with -a only)
-f             force adding or deleting of replicas
-k filename    alternate /etc/system file
-l length      specify size of replica (for use with -a only)
# /usr/opt/SUNWmd/sbin/metadb
    flags        first blk       block count
    a m  p  lu       16           1034         /dev/dsk/c0t3d0s3
    a    p  l        1050         1034         /dev/dsk/c0t3d0s3
```

4. **Reboot the system.**

```
# reboot
rebooting...
```

5. **Once you have a replacement disk, halt the system, replace the failed disk, and once again, reboot the system. Use the `format` command to partition the disk as it was before the failure.**

```
# halt
...
boot
...
# format /dev/rdsk/c1t2d0s0
...
```

6. **Use the `metadb` command to add back the database replicas and to determine that the replicas are correct.**

```
# /usr/opt/SUNWmd/sbin/metadb -c 2 -a /dev/dsk/c1t2d0s3
# /usr/opt/SUNWmd/sbin/metadb
   flags        first blk  block count
  a m  p  luo    16         1034                /dev/dsk/c0t3d0s3
  a    p  luo    1050       1034                /dev/dsk/c0t3d0s3
  a       u      16         1034                /dev/dsk/c1t2d0s3
  a       u      1050       1034                /dev/dsk/c1t2d0s3
```

7. **Use the `metareplace` command to re-enable the submirrors.**

```
# /usr/opt/SUNWmd/sbin/metareplace -e d0 /dev/dsk/c1t2d0s0
Device /dev/dsk/c1t2d0s0 is enabled

# /usr/opt/SUNWmd/sbin/metareplace -e d1 /dev/dsk/c1t2d0s1
Device /dev/dsk/c1t2d0s1 is enabled

# /usr/opt/SUNWmd/sbin/metareplace -e d2 /dev/dsk/c1t2d0s6
Device /dev/dsk/c1t2d0s6 is enabled
```

The submirrors will now resync.

C

Boot Device Fails

If your boot device fails, you'll need to set up an alternate boot device. In the following example, the boot device containing two of the six database replicas and the `root`, `swap`, and `/usr` submirrors fails. The basic procedure is to repair the disk, boot from another root submirror, and then restore the database and mirrors to their original state.

Initially, when the boot device fails, you'll see a message similar to the following. This message may differ among various architectures.

```
Booting from: sd(0,0,0)/kernel/unix
The selected SCSI device is not responding
Can't open boot device
...
```

When you see this message, it would be a very good idea to make a note of the device. Then, follow these steps:

1. **Boot from another `root` submirror.**
   Since only two of the six database replicas in this example are in error, you can still boot. If this were not the case, you would need to delete the database replicas in single-user mode. This procedure is described in "Stale Metadevice Database" on page 234.

Note – Having database replicas on at least three disks would make this procedure unnecessary in the case of a single disk failure.

```
ok boot sd(0,2,0)
Booting from: sd(0,2,0)
...
Copyright (c) 1983-1992, Sun Microsystems, Inc.
Hostname: demo
...
demo console login: root
Password:
Last login: Wed Dec 16 13:15:42 on console
SunOS Release 5.1 Version Generic [UNIX(R) System V Release 4.0]
...
#
```

2. **Use the `metadb` command to determine that two database replicas have failed.**

```
# /usr/opt/SUNWmd/sbin/metadb
       flags           first blk       block count
    M     p            unknown         unknown         /dev/dsk/c0t3d0s3
    M     p            unknown         unknown         /dev/dsk/c0t3d0s3
    a m   p  luo       16              1034            /dev/dsk/c0t2d0s3
    a     p  luo       1050            1034            /dev/dsk/c0t2d0s3
    a     p  luo       16              1034            /dev/dsk/c0t1d0s3
    a     p  luo       1050            1034            /dev/dsk/c0t1d0s3
```

3. **Use the `metastat` command to determine that half of the `root`, `swap`, and `/usr` mirrors have failed.**

```
# /usr/opt/SUNWmd/sbin/metastat
d0: Mirror
    Submirror 0: d10
      State: Needs maintenance
    Submirror 1: d20
      State: Okay
    Pass: 1
    Read option: roundrobin (default)
    Write option: parallel (default)
    Size: 47628 blocks

d10: Submirror of d0
    State: Needs maintenance
    Invoke: "metareplace d0 /dev/dsk/c0t3d0s0 <new device>"
    Size: 47628 blocks
    Stripe 0:
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c0t3d0s0          0     No    Maintenance

d20: Submirror of d0
    State: Okay
    Size: 47628 blocks
    Stripe 0:
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c0t2d0s0          0     No    Okay
```

```
(continued from previous page)

d1: Mirror
    Submirror 0: d11
      State: Needs maintenance
    Submirror 1: d21
      State: Okay
    Pass: 2
    Read option: roundrobin (default)
    Write option: parallel (default)
    Size: 69660 blocks

d11: Submirror of d1
    State: Needs maintenance
    Invoke: "metareplace d1 /dev/dsk/c0t3d0s1 <new device>"
    Size: 69660 blocks
    Stripe 0:
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c0t3d0s1          0     No    Maintenance

d21: Submirror of d1
    State: Okay
    Size: 69660 blocks
    Stripe 0:
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c0t2d0s1          0     No    Okay

d2: Mirror
    Submirror 0: d12
      State: Needs maintenance
    Submirror 1: d22
      State: Okay
    Pass: 3
    Read option: roundrobin (default)
    Write option: parallel (default)
    Size: 286740 blocks
```

```
(continued from previous page)

d2: Mirror
    Submirror 0: d12
      State: Needs maintenance
    Submirror 1: d22
      State: Okay
    Pass: 3
    Read option: roundrobin (default)
    Write option: parallel (default)
    Size: 286740 blocks

d12: Submirror of d2
    State: Needs maintenance
    Invoke: "metareplace d2 /dev/dsk/c0t3d0s6 <new device>"
    Size: 286740 blocks
    Stripe 0:
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c0t3d0s6          0     No    Maintenance


d22: Submirror of d2
    State: Okay
    Size: 286740 blocks
    Stripe 0:
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c0t2d0s6          0     No    Okay
```

4. **Halt the system, repair the disk, and reboot.**
   Note that you must reboot from the other half of the root mirror.

```
# halt
...
Halted
...
ok boot sd(0,2,0)
Booting from: sd(0,2,0)
...
SunOS Release 5.1 Version Generic [UNIX(R) System V Release 4.0]
Copyright (c) 1983-1992, Sun Microsystems, Inc.
...
Hostname: demo
The system is coming up.  Please wait.
...
The system is ready.

demo console login: root
Password:
Last login: Wed Dec 16 13:36:29 on console
SunOS Release 5.1 Version Generic [UNIX(R) System V Release 4.0]
#
```

5. **Use the `metadb` command to delete the failed replicas and then add them back.**

```
# /usr/opt/SUNWmd/sbin/metadb
       flags           first blk       block count
    M     p            unknown         unknown         /dev/dsk/c0t3d0s3
    M     p            unknown         unknown         /dev/dsk/c0t3d0s3
    a m   p  luo       16              1034            /dev/dsk/c0t2d0s3
    a     p  luo       1050            1034            /dev/dsk/c0t2d0s3
    a     p  luo       16              1034            /dev/dsk/c0t1d0s3
    a     p  luo       1050            1034            /dev/dsk/c0t1d0s3
# /usr/optSUNWmd/sbin/metadb -d c0t3d0s3
# /usr/optSUNWmd/sbin/metadb -c2 -a /dev/dsk/c0t3d0s3
# /usr/optSUNWmd/sbin/metadb
       flags           first blk       block count
     a        u        16              1034            /dev/dsk/c0t3d0s3
     a        u        1050            1034            /dev/dsk/c0t3d0s3
     a m   p  luo      16              1034            /dev/dsk/c0t2d0s3
     a     p  luo      1050            1034            /dev/dsk/c0t2d0s3
     a     p  luo      16              1034            /dev/dsk/c0t1d0s3
     a     p  luo      1050            1034            /dev/dsk/c0t1d0s3
```

6. **Use the `metareplace` command to re-enable the submirrors.**

```
# /usr/opt/SUNWmd/sbin/metareplace -e d0 /dev/dsk/c0t3d0s0
Device /dev/dsk/c0t3d0s0 is enabled

# /usr/opt/SUNWmd/sbin/metareplace -e d1 /dev/dsk/c0t3d0s1
Device /dev/dsk/c0t3d0s1 is enabled

# /usr/opt/SUNWmd/sbin/metareplace -e d2 /dev/dsk/c0t3d0s6
Device /dev/dsk/c0t3d0s6 is enabled
```

After some time, the resyncs will complete. You can now return to booting from the original device.

At this point, running the `metastat` command would display the following:

```
# /usr/opt/SUNWmd/sbin/metastat
d0: Mirror
    Submirror 0: d10
      State: Okay
    Submirror 1: d20
      State: Okay
    Pass: 1
    Read option: roundrobin (default)
    Write option: parallel (default)
    Size: 47628 blocks

d10: Submirror of d0
    State: Okay
    Size: 47628 blocks
    Stripe 0:
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c0t3d0s0          0     No    Okay

d20: Submirror of d0
    State: Okay
    Size: 47628 blocks
    Stripe 0:
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c0t2d0s0          0     No    Okay

d1: Mirror
    Submirror 0: d11
      State: Okay
    Submirror 1: d21
      State: Okay
    Pass: 2
    Read option: roundrobin (default)
    Write option: parallel (default)
    Size: 69660 blocks
```

```
(continued from previous page)

d11: Submirror of d1
    State: Okay
    Size: 69660 blocks
    Stripe 0:
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c0t3d0s1           0     No    Okay

d21: Submirror of d1
    State: Okay
    Size: 69660 blocks
    Stripe 0:
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c0t2d0s1           0     No    Okay

d2: Mirror
    Submirror 0: d12
      State: Okay
    Submirror 1: d22
      State: Okay
    Pass: 3
    Read option: roundrobin (default)
    Write option: parallel (default)
    Size: 286740 blocks

d12: Submirror of d2
    State: Okay
    Size: 286740 blocks
    Stripe 0:
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c0t3d0s6           0     No    Okay

d22: Submirror of d2
    State: Okay
    Size: 286740 blocks
    Stripe 0:
    Device              Start Block  Dbase State        Hot Spare
    /dev/dsk/c0t2d0s6           0     No    Okay
```

C

Upgrading to Other Solaris Versions D

Introduction

Upgrading to later versions of the Solaris environment while using metadevices requires steps not currently outlined in the Solaris documentation. The current Solaris upgrade procedure is incompatible with DiskSuite. The following supplemental procedure is provided as an alternative to completely reinstalling the Solaris and DiskSuite packages.

Note – You must have the media to upgrade Solaris (and DiskSuite if necessary).

Upgrading Solaris With Solstice DiskSuite

Warning – Before you begin this procedure, back up all file systems. See the `ufsdump(1M)` manual page for details.

1. **Repair any mirrors that have errors.**
2. **Save `/etc/vfstab` for later use.**

3. **Clear any metatrans devices that may be used during the Solaris upgrade (for example, `/usr`, `/var`, and `/opt`).**
   See Chapter 6, "UFS Logging" for information on clearing (removing logging from) metatrans devices. If you are uncertain which metatrans devices should be cleared, clear all metatrans devices.

4. **Comment out file systems in `/etc/vfstab` mounted on metadevices that are not simple metadevices or simple mirrors.**
   A simple metadevice is composed of a single component with a `Start Block` of 0. A simple mirror is composed of submirrors, all of which are simple metadevices.

5. **Convert the remaining (simple) mirrors to one-way mirrors with the `metadetach` command.**
   Upgrade will be performed on a single submirror of each mirror. The other submirrors will be synced up with `metattach` after the upgrade.

6. **If `root` is mounted on a metadevice or mirror, set the `root` file system to be mounted on the underlying component of the metadevice or the underlying component of the remaining attached submirror.**
   Use the `metaroot` command to do this safely.

7. **Edit the `/etc/vfstab` file to change any file systems or `swap` devices still mounted on metadevices or mirrors after Step 3.**
   Mount the file systems on the underlying component of the metadevices or the underlying component of the remaining attached submirrors

8. **Remove symbolic links to the DiskSuite startup files so that it is no longer initialized at boot time.**

```
demo# rm /etc/rcS.d/S35SUNWmd.init /etc/rc2.d/S95SUNWmd.sync
```

   These links will be added back later by reinstalling DiskSuite after the Solaris upgrade.

9. **Halt the machine and upgrade Solaris, then reboot the machine.**

10. **Reinstall DiskSuite, then reboot the machine.**
    This will re-establish the symbolic links removed in Step 7.

Note – Make certain that the version of Solaris you are installing is compatible with DiskSuite 4.0.

11. **If `root` was originally mounted on a metadevice or mirror, set the `root` file system to be mounted back on the original metadevice or mirror.** Use the `metaroot` command to do this safely.
12. **Edit the `/etc/vfstab` file to change any file systems or `swap` devices edited in Step 6 to be mounted back on their original metadevice or mirror.**
13. **Edit the `/etc/vfstab` file to uncomment the file systems commented out in Step 3.**
14. **Reboot the machine to remount the file systems.**
15. **Use the `metattach` command to reattach and resync any submirrors broken off in Step 4.**
16. **Recreate the cleared metatrans devices. See Chapter 6, "UFS Logging" for information on creating metatrans devices.**

D

Using Solstice DiskSuite 4.0 with the SPARCstorage Array 100

Solstice DiskSuite 4.0 is fully compatible with SPARCstorage Array 100 (SSA-100). In most situations, using SSA-100 disks with DiskSuite is just like using any other disks. There are, however, some characteristics of the SSA-100 of which DiskSuite users should be aware. The following sections detail these characteristics and explain how DiskSuite should be configured and used to take advantage of the SSA-100. This appendix assumes that you have Version 2.0 of the SPARCstorage Array software.

For situations not covered in this appendix, please refer to the *SPARCstorage Array User's Guide* for more information.

Installation

The SSA-100 should be installed according to the instructions in the *SSA User's Guide*. In particular, when using Solaris 2.4 or earlier releases, the SSA drivers contained in the SSA software distribution must be installed prior to using the SSA-100 with DiskSuite. The SSA Volume Manager need not be installed if you are only using DiskSuite.

E

SSA Specific Operations

Solstice DiskSuite 4.0 does not currently feature a graphical interface for accessing the SSA specific operations. To use these operations, you must use the SSA command line interface (CLI) provided with the SSA software distribution. Please refer to the sections describing SSA specific CLI operations in the *SSA User's Guide* for instructions on how to use these commands.

Device Naming

DiskSuite accesses SSA-100 disks exactly like any other disks, with one important exception: the disk names differ from non-SSA disks.

The SSA-100 disk naming convention is:

`c[0-n]t[0-5]d[0-4]s[0-7]`

where:

- `c` indicates the controller attached to an SSA unit
- `t` indicates one of the 6 SCSI strings within an SSA
- `d` indicates one of the 5 disks on an internal SCSI string
- `s` indicates the disk slice number
- Strings `t0` and `t1` are contained in tray 1, t2 and `t3` in tray 2, and `t4` and `t5` are in tray 3

Configuring for Online Replacement

If you want to take advantage of the SSA-100's online replacement capabilities, you must use either mirrors or RAID metadevices in order to provide data redundancy. The following sections describe how to set up mirrors and RAID devices so that failed drives can be replaced without loss of service.

Mirrors

When setting up mirrors on an SSA-100, you should configure the submirrors within a mirror from disks in different trays. This way, if a disk within a submirror fails, it will be possible to remove the tray and replace the drive without impacting the other submirror.

For example, consider a two-way mirror with each submirror composed of a concatenation of three SSA-100 disks. The following commands initialize this mirror using the first three drives from tray 1 and the first three drives in tray 2.

```
# metainit d1 3 1 c0t0d0s2 1 c0t0d1s2 1 c0t0d2s2
# metainit d2 3 1 c0t2d0s2 1 c0t2d1s2 1 c0t2d2s2
# metainit d0 -m d1
# metattach d0 d2
```

Replacing a Disk Drive in a Mirror

Use the following steps to replace a SPARCstorage Array 100 drive when using DiskSuite. Where appropriate, the instructions give the DiskSuite CLI command required to perform the task. Each of these steps may also be performed using DiskSuite Tool. In this case, use the DiskSuite Tool operation corresponding to the given CLI command. You may want to consult the *DiskSuite Tool User's Guide* if you are unsure of which graphical operation corresponds to a given command.

Note – The following procedure is for Solaris 2.4 and previous Solaris versions only.

1. **Use the `metaoffline` DiskSuite command to instruct DiskSuite to stop using all drives in the tray containing the failed drive.**
   This may require offlining drives in submirrors unrelated to the submirror being repaired.

Warning – Any applications using non-replicated disks in this tray should now be suspended or terminated.

2. **Use the `ssacli sync_cache <ctlr>` SSA command to flush outstanding SSA writes in NVRAM.**
3. **Use the `ssacli -t<1|2|3> stop <ctlr>` SSA command to spin down all drives in the tray containing the failed drive.**
4. **Remove the tray containing the failed drive and replace the drive.**

5. **Use the** `ssacli -t<1|2|3> start <ctlr>` SSA **command to spin up the drives in the tray.**
6. **Using** `format(1M)` **or** `fmthard(1M)`**, correctly partition the new drive.**
7. **Use the** `metaonline` **DiskSuite command to instruct DiskSuite to start using the drives in the tray again.**
8. **Use the** `metareplace -e` **command to enable the new drive.**

RAID-5 Metadevices

When setting up RAID-5 metadevices for online repair, you will have to use a minimum RAID-5 width of 3 components. While this is not an optimal configuration for RAID-5, it is still slightly less expensive than mirroring, in terms of the overhead of the redundant data. You should place each of the 3 components of each RAID-5 metadevice within a separate tray. If all disks in an SSA-100 are configured this way (or in combination with mirrors as described above), the tray containing the failed component may be removed without losing access to any of the data.

Replacing a Disk Drive in a RAID-5 Metadevice

Warning – Any applications using non-replicated disks in the tray containing the failed drive should first be suspended or terminated.

1. **Use the** `ssacli sync_cache <ctlr>` SSA **command to flush outstanding SSA writes in NVRAM.**
2. **Use the** `ssacli -t<1|2|3> stop <ctlr>` SSA **command to spin down all drives in the tray containing the failed drive.**
3. **Remove the tray containing the failed drive and replace the drive.**
4. **Use the** `ssacli -t<1|2|3> start <ctlr>` SSA **command to spin up the drives in the tray.**
5. **Using** `format(1M)` **or** `fmthard(1M)`**, correctly partition the new drive.**
6. **Use the** `metareplace -e` **command to enable the new drive in the tray.**

Man Pages

F

Introduction

This appendix contains the manual pages associated with DiskSuite in the format of the *SunOS Reference Manual*. The manual pages included in this appendix are:

`md.tab(4)` – metadisk configuration file

`mddb.cf(4)` - metadevice state database replica locations

`md(7)` - user configurable pseudo device driver for metadevices

`growfs(1M)` – nondestructively expand a mounted file system

`metaclear(1M)` – clear active metadevices

`metadb(1M)` – create and delete replicas of the metadevice state database

`metahs(1M)` – manage metadisk hot spares and hot spare pools

`metainit(1M)` – configure metadevices specified in `md.tab`

`metaoffline(1M)` – offline and online submirrors

`metaparam(1M)` – modify parameters of metadevices and metamirrors

`metareplace(1M)` – replace components of submirrors

`metaroot (1M)` – setup system files for root metadevice

F

`metaset(1M)` – configure metadevice disksets

`metastat(1M)` – print status for metadevice

`metasync(1M)` – handle mirror resync during reboot

`metattach(1M)` – attach or detach metadevice to or from metamirror

Note – The manual pages described above are included only in the printed version of this document. These manual pages are not available when you use the AnswerBook online documentation to view this manual.

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W