VERITAS™

# VERITAS File System 4.1

## Administrator's Guide

**HP-UX**

N11865G

June 2005

## Disclaimer

The information contained in this publication is subject to change without notice. VERITAS Software Corporation makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. VERITAS Software Corporation shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual.

## VERITAS Legal Notice



VERITAS Software Corporation
350 Ellis Street
Mountain View, CA 94043
USA
Phone 650–527–8000 Fax 650–527–2908
www.veritas.com

## Third-Party Legal Notices

### Data Encryption Standard (DES) Copyright

# Contents





# Preface

The *VERITAS File System Administrator's Guide* provides information on the most important aspects of VERITAS File System (VxFS) administration. This guide is for system administrators who configure and maintain UNIX systems with the VERITAS File System, and assumes that you have a:

- Basic understanding of system administration
- Working knowledge of the UNIX operating system
- General understanding of file systems



# How This Guide Is Organized

Chapter 1. "The VERITAS File System" on page 1 introduces the major features and characteristics of VxFS.

Chapter 2. "VxFS Performance: Creating, Mounting, and Tuning File Systems" on page 21 describes VxFS tools that optimize system performance. This section includes information on mount options.

Chapter 3. "Extent Attributes" on page 47 describes the policies associated with allocation of disk space.

Chapter 4. "Application Interface" on page 53 describes ways to optimize an application for use with VxFS. This chapter includes details on cache advisories, extent sizes, and reservation of file space.

Chapter 5. "Storage Checkpoints" on page 65 describes the VxFS replication technology that allows the quick and easy creation of resource-efficient file system backups.

Chapter 6. "Online Backup Using File System Snapshots" on page 89 describes the snapshot backup feature of VxFS.

Chapter 7. "Quotas" on page 97 describes VxFS methods to limit user access to file and data resources.

Chapter 8. "File Change Log" on page 101 describes the File Change Log and Reverse Name Lookup feature of VxFS.

Chapter 9. "Multi-Volume File Systems" on page 107 describes the multi-volume support feature that allows several volumes to be represented by a single logical object in a volume set.

Chapter 10. "Quality of Storage Service" on page 115 describes the VERITAS Quality of Storage Service option that lets VxFS map more than one volume into a single file system.

Chapter 11. "Quick I/O for Databases" on page 123 describes the VERITAS Quick I/O™ feature that treats preallocated files as raw character devices to increase performance.

Appendix A. "VERITAS File System Quick Reference" on page 135 provides information on common file system tasks and examples of typical VxFS operations.

Appendix B. "Kernel Messages" on page 163 lists VxFS kernel error messages in numerical order and provides explanations and suggestions for dealing with these problems.

Appendix C. "Disk Layout" on page 205 describes and illustrates the major components of VxFS disk layouts.

The "Glossary" contains a list of terms and definitions relevant to VxFS.

# Conventions

| Typeface | Usage | Examples |
|---|---|---|
| `monospace` | Computer output, files, directories, software elements such as command options, function names, and parameters | Read tunables from the `/etc/vx/tunefstab` file.<br>See the `vxtunefs`(1M) manual page for more information. |
| **`monospace`** (**`bold`**) | User input | `#` **`mount -F vxfs /h/filesys`** |
| *italic* | New terms, book titles, emphasis, variables replaced with a name or value | See the *User's Guide* for details.<br>The variable *vxfs_ninode* determines the value of... |

| Symbol | Usage | Examples |
|---|---|---|
| % | C shell prompt | |
| $ | Bourne/Korn/Bash shell prompt | |
| # | Superuser prompt (all shells) | |
| \ | Continued input on the following line; you do not type this character | `#` **`mount -F vxfs`** `\`<br>**`/h/filesys`** |
| `[ ]` | In a command synopsis, brackets indicates an optional argument | `ls [-a]` |
| `\|` | In a command synopsis, a vertical bar separates mutually exclusive arguments | `mount [suid \| nosuid ]` |
| blue text | Indicates an active hypertext link | In PDF and HTML files, click on links to move to the specified location |



# Getting Help

For technical assistance, visit http://support.veritas.com and select phone or email support. This site also provides access to resources such as TechNotes, product alerts, software downloads, hardware compatibility lists, and the VERITAS customer email notification service. Use the Knowledge Base Search feature to access additional product information, including current and past releases of product documentation.

Diagnostic tools are also available to assist in troubleshooting problems associated with the product. These tools are available on disc or can be downloaded from the VERITAS FTP site. See the `README.VRTSspt` file in the `/support` directory for details.

For license information, software updates and sales contacts, visit https://my.veritas.com/productcenter/ContactVeritas.jsp. For information on purchasing product documentation, visit http://webstore.veritas.com.

# The VERITAS File System 1

VxFS is an extent-based, intent logging file system. VxFS is designed for use in UNIX environments that require high performance and availability and deal with large amounts of data.

This chapter provides an overview of major VxFS features that are described in detail in later chapters. The following topics are introduced in this chapter:





- Multi-Volume Support
- Quality of Storage Service

# VxFS Features

Basic features include:

- Extent-based allocation
- Extent attributes
- Fast file system recovery
- Access control lists (ACLs)
- Online administration
- Online backup
- Enhanced application interface
- Enhanced mount options
- Improved synchronous write performance
- Support for file systems up to 32 terabytes in size
- Support for files up to 2 terabytes in size
- Enhanced I/O performance
- Quotas
- Cluster File System
- Improved database performance
- Storage Checkpoints
- Cross-platform data sharing
- File Change Log
- Multi-volume support
- Quality of Storage Service

**Note** VxFS supports all HFS file system features and facilities except for the linking, removing, or renaming of "`.`" and "`..`" directory entries. Such operations may disrupt file system sanity.



# Disk Layouts

The disk layout is the way file system information is stored on disk. On VxFS, six disk layout versions, numbered 1 through 6, were created to support various new features and specific UNIX environments. Currently, only the Version 4, 5, and 6 disk layouts can be created and mounted.

See "Disk Layout" on page 205 for a description of the disk layouts.

# File System Performance Enhancements

Traditional file systems employ block-based allocation schemes that provide adequate random access and latency for small files, but which limit throughput for larger files. As a result, they are less than optimal for commercial environments.

VxFS addresses this file system performance issue through an alternative allocation method and increased user control over allocation, I/O, and caching policies. An overview of the VxFS allocation policy is provided in the section "Extent-Based Allocation" on page 6.

VxFS provides the following performance enhancements:

- Extent-based allocation
- Enhanced mount options
- Data synchronous I/O
- Direct I/O and discovered direct I/O
- Caching advisories
- Enhanced directory features
- Explicit file alignment, extent size, and preallocation controls
- Tunable I/O parameters
- Tunable indirect data extent size
- Integration with VERITAS Volume Manager™ (VxVM®)
- Support for improved database performance

The rest of this chapter, as well as "VxFS Performance: Creating, Mounting, and Tuning File Systems" on page 21 and "Application Interface" on page 53 provide details on many of these features.

# VERITAS Enterprise Administrator Graphical User Interface

The VERITAS Enterprise Administrator (VEA) is a GUI based on the Java™ technology that consists of a server and a client. The server runs on a UNIX system that is running the VERITAS Volume Manager and VxFS. The client runs on any platform that supports the Java Runtime Environment. You can use VEA to perform a subset of VxFS administrative functions on a local or remote system. These functions include:

- Creating a New File System on a Volume
- Creating a New File System on a Volume Set
- Removing a File System from the File System Table
- Mounting/Unmounting a File System
- Defragmenting a File System
- Monitoring File System Capacity
- Creating a Snapshot Copy of a File System
- Checking a File System
- Viewing File System Properties
- Using the QuickLog Feature
- Maintaining the File Change Log
- Maintaining Storage Checkpoints
- Using Multi-Volume File Systems
- Setting Intent Log Options
- Unmounting a File System from a Cluster Node
- Removing Resource Information for a Cluster File System

For instructions on how to use VEA, see the *VERITAS Volume Manager User's Guide – VERITAS Enterprise Administrator*.



# Extent-Based Allocation

Disk space is allocated in 1024-byte sectors to form logical blocks. VxFS supports logical block sizes of 1024, 2048, 4096, and 8192 bytes. The default block size is 1K.

An *extent* is defined as one or more adjacent blocks of data within the file system. An extent is presented as an *address-length* pair, which identifies the starting block address and the length of the extent (in file system or logical blocks). VxFS allocates storage in groups of extents rather than a block at a time.

Extents allow disk I/O to take place in units of multiple blocks if storage is allocated in consecutive blocks. For sequential I/O, multiple block operations are considerably faster than block-at-a-time operations; almost all disk drives accept I/O operations of multiple blocks.

Extent allocation only slightly alters the interpretation of addressed blocks from the inode structure compared to block based inodes. A VxFS inode references 10 direct extents, each of which are pairs of starting block addresses and lengths in blocks. The VxFS inode also points to two indirect address extents, which contain the addresses of other extents:

- The first indirect address extent is used for single indirection; each entry in the extent indicates the starting block number of an indirect data extent.
- The second indirect address extent is used for double indirection; each entry in the extent indicates the starting block number of a single indirect address extent.

Each indirect address extent is 8K long and contains 2048 entries. All indirect data extents for a file must be the same size; this size is set when the first indirect data extent is allocated and stored in the inode. Directory inodes always use an 8K indirect data extent size. By default, regular file inodes also use an 8K indirect data extent size that can be altered with `vxtunefs` (see "Tuning VxFS I/O Parameters" on page 37); these inodes allocate the indirect data extents in clusters to simulate larger extents.

## Typed Extents

VxFS has an inode block map organization for indirect extents known as *typed extents*. Each entry in the block map has a typed descriptor record containing a type, offset, starting block, and number of blocks.

Indirect and data extents use this format to identify logical file offsets and physical disk locations of any given extent. The extent descriptor fields are defined as follows:

| | |
|---|---|
| type | Uniquely identifies an extent descriptor record and defines the record's length and format. |
| offset | Represents the logical file offset in blocks for a given descriptor. Used to optimize lookups and eliminate hole descriptor entries. |
| starting block | The starting file system block of the extent. |
| number of blocks | The number of contiguous blocks in the extent. |

- Indirect address blocks are fully typed and may have variable lengths up to a maximum and optimum size of 8K. On a fragmented file system, indirect extents may be smaller than 8K depending on space availability. VxFS always tries to obtain 8K indirect extents but resorts to smaller indirects if necessary.
- Indirect Data extents are variable in size to allow files to allocate large, contiguous extents and take full advantage of VxFS's optimized I/O.
- Holes in sparse files require no storage and are eliminated by typed records. A hole is determined by adding the offset and length of a descriptor and comparing the result with the offset of the next record.
- While there are no limits on the levels of indirection, lower levels are expected in this format since data extents have variable lengths.
- This format uses a type indicator that determines its record format and content and accommodates new requirements and functionality for future types.

The current typed format is used on regular files and directories only when indirection is needed. Typed records are longer than the previous format and require less direct entries in the inode. Newly created files start out using the old format which allows for ten direct extents in the inode. The inode's block map is converted to the typed format when indirection is needed to offer the advantages of both formats.



# Extent Attributes

VxFS allocates disk space to files in groups of one or more extents. VxFS also allows applications to control some aspects of the extent allocation. *Extent attributes* are the extent allocation policies associated with a file.

The `setext` and `getext` commands allow the administrator to set or view extent attributes associated with a file, as well as to preallocate space for a file. Refer to "Extent Attributes" on page 47, "Application Interface" on page 53, and the `setext`(1M) and `getext`(1M) manual pages for discussions on how to use extent attributes.

The `vxtunefs` command allows the administrator to set or view the default indirect data extent size. Refer to "VxFS Performance: Creating, Mounting, and Tuning File Systems" on page 21 and the `vxtunefs`(1M) manual page for discussions on how to use the indirect data extent size feature.

# Fast File System Recovery

Most file systems rely on full structural verification by the `fsck` utility as the only means to recover from a system failure. For large disk configurations, this involves a time-consuming process of checking the entire structure, verifying that the file system is intact, and correcting any inconsistencies.

## VxFS Intent Log

VxFS reduces system failure recovery times by tracking file system activity in the VxFS *intent log*. This feature records pending changes to the file system structure in a circular *intent log*. The intent log recovery feature is not readily apparent to users or a system administrator except during a system failure. During system failure recovery, the VxFS `fsck` utility performs an intent log replay, which scans the intent log and nullifies or completes file system operations that were active when the system failed. The file system can then be mounted without completing a full structural check of the entire file system. Replaying the intent log may not completely recover the damaged file system structure if there was a disk hardware failure; hardware problems may require a complete system check using the `fsck` utility provided with VxFS.

## Intent Log Resizing

The VxFS intent log is allocated when the file system is first created. The size of the intent log is based on the size of the file system—the larger the file system, the larger the intent log. The maximum default intent log size for disk layout Versions 4, 5, and 6 is 16 megabytes.

With the Version 6 disk layout, you can dynamically increase or decrease the intent log size using the `logsize` option of the `fsadm` command. Increasing the size of the intent log can improve system performance because it reduces the number of times the log wraps around. However, increasing the intent log size can lead to greater times required for a log replay if there is a system failure.

**Note** Inappropriate sizing of the intent log can have a negative impact on system performance.

See the `mkfs_vxfs`(1M) and the `fsadm_vxfs`(1M) manual pages for more information on intent log size.

# Online System Administration

A VxFS file system can be defragmented and resized while it remains online and accessible to users. The following sections provide an overview of these features.

## Defragmentation

Free resources are initially aligned and allocated to files in an order that provides optimal performance. On an active file system, the original order of free resources is lost over time as files are created, removed, and resized. The file system is spread farther along the disk, leaving unused gaps or *fragments* between areas that are in use. This process is also known as *fragmentation* and leads to degraded performance because the file system has fewer options when assigning an 3 extent to a file (a group of contiguous data blocks).

VxFS provides the online administration utility `fsadm` to resolve the problem of fragmentation. The `fsadm` utility defragments a mounted file system by:

- Removing unused space from directories.
- Making all small files contiguous.
- Consolidating free blocks for file system use.

This utility can run on demand and should be scheduled regularly as a `cron` job.



## File System Resizing

A file system is assigned a specific size as soon as it is created; the file system may become too small or too large as changes in file system usage take place over time.

Most large file systems with too much space try to reclaim the unused space by off-loading the contents of the file system and rebuilding it to a preferable size. The VxFS utility `fsadm` can expand or shrink a file system without unmounting the file system or interrupting user productivity. However, to expand a file system, the underlying device on which it is mounted must be expandable.

VxVM facilitates expansion using virtual disks that can be increased in size while in use. The VxFS and VxVM packages complement each other to provide online expansion capability. Use the `vxresize` command when resizing both the volume and the file system. The `vxresize` command guarantees that the file system will shrink or grow along with the volume. Do not use the `vxassist` and `fsadm_vxfs` commands for this purpose. See the `vxresize`(1M) manual page and the *VERITAS Volume Manager Administrator's Guide* for more information.

# Application Interface

VxFS conforms to the System V Interface Definition (SVID) requirements and supports user access through the Network File System (NFS). Applications that require performance features not available with other file systems can take advantage of VxFS enhancements that are introduced in this section and covered in detail in "Application Interface" on page 53.

## Application Transparency

In most cases, any application designed to run on native file systems will run transparently on VxFS.

### Expanded Application Facilities

VxFS provides some facilities frequently associated with commercial applications that make it possible to:

- Preallocate space for a file.
- Specify a fixed extent size for a file.
- Bypass the system buffer cache for file I/O.
- Specify the expected access pattern for a file.

Because these facilities are provided using VxFS-specific ioctl system calls, most existing UNIX system applications do not use them. The `cp`, `cpio`, and `mv` utilities use the facilities to preserve extent attributes and allocate space more efficiently. The current attributes of a file can be listed using the `getext` command or `ls` command. The facilities can also improve performance for custom applications. For portability reasons, these applications must check which file system type they are using before using these interfaces.

## Extended mount Options

The VxFS file system supports extended `mount` options to specify:

- Enhanced data integrity modes.
- Enhanced performance modes.
- Temporary file system modes.
- Improved synchronous writes.
- Large file sizes.

See "VxFS Performance: Creating, Mounting, and Tuning File Systems" on page 21 and the `mount_vxfs`(1M) manual page for details on the VxFS `mount` options.



## Enhanced Data Integrity Modes

**Note** There are performance advantages and disadvantages associated with the use of these `mount` options.

For most UNIX file systems, including VxFS, the default mode for writing to a file is delayed, or buffered, meaning that the data to be written is copied to the file system cache and later flushed to disk in a lazy fashion.

This provides much better performance than synchronously writing the data to disk. However, in the event of a system failure, data written shortly before the failure may be lost since it was not flushed to disk. In addition, if space was allocated to the file as part of the write request, and the corresponding data was not flushed to disk before the system failure occurred, unitialized data can appear in the file.

For the most common type of write, delayed extending writes (a delayed write that increases the file size), VxFS avoids the problem of unitialized data appearing in the file by waiting until the data has been flushed to disk before updating the new file size to disk. If a system failure occurs before the data has been flushed to disk, the file size has not yet been updated to be unitialized data, thus no uninitialized data appears in the file. The unused blocks that were allocated are reclaimed.

### Using blkclear Option for Data Integrity

In environments where performance is more important than absolute data integrity, the preceding situation is not of great concern. However, VxFS supports environments that emphasize data integrity by providing the `mount -o blkclear` option that ensures uninitialized data does not appear in a file.

### Using closesync Option for Data Integrity

VxFS provides the `mount -o mincache=closesync` option, which is useful in desktop environments with users who are likely to shut off the power on machines without halting them first. In `closesync` mode, only files that are written during the system crash or shutdown can lose data. Any changes to a file are flushed to disk when the file is closed.

### Using the log Option for Data Integrity

File systems are typically asynchronous in that structural changes to the file system are not immediately written to disk, which provides better performance. However, recent changes made to a system can be lost if a system failure occurs. Specifically, attribute changes to files and recently created files may disappear.

The `mount -o log` intent logging option guarantees that all structural changes to the file system are logged to disk before the system call returns to the application. With this option, the `rename`(2) system call flushes the source file to disk to guarantee the persistent of the file data before renaming it. The `rename`() call is also guaranteed to be persistent when the system call returns. The changes to file system data and metadata caused by the `fsync`(2) and `fdatasync`(2) system calls are guaranteed to be persistent once the calls return.

## Enhanced Performance Mode

VxFS has several mount options that improve performance, such as `delaylog`.

### Using the delaylog Option for Enhanced Performance

The default VxFS logging mode, `mount -o delaylog`, increases performance by delaying the logging of some structural changes, but does not provide the equivalent data integrity as the previously described modes. That is because recent changes may be lost during a system failure. This option provides at least the same level of data accuracy that traditional UNIX file systems provide for system failures, along with fast file system recovery. `delaylog` is the default `mount` option.



## Modes of Temporary File System

On most UNIX systems, temporary file system directories (such as `/tmp` and `/usr/tmp`) often hold files that do not need to be retained when the system reboots. The underlying file system does not need to maintain a high degree of structural integrity for these temporary directories.

### Using the tmplog option For Temporary File Systems

VxFS provides a `mount -o tmplog` option, which allows the user to achieve higher performance on temporary file systems by delaying the logging of most operations.

## Improved Synchronous Writes

VxFS provides superior performance for synchronous write applications. The default `mount -o datainlog` option greatly improves the performance of small synchronous writes.

The `mount -o convosync=dsync` option improves the performance of applications that require synchronous data writes but not synchronous inode time updates.

**Caution** The use of the `-o convosync=dsync` option violates POSIX semantics.

## Support for Large Files

VxFS can support files larger than two terabytes. See "largefiles | nolargefiles" on page 29 for information on how to create, mount, and manage file systems containing large files.

**Caution** Some applications and utilities may not work on large files.

# Enhanced I/O Performance

VxFS provides enhanced I/O performance by applying an aggressive I/O clustering policy, integrating with VxVM, and allowing application specific parameters to be set on a per-file system basis.

## Enhanced I/O Clustering

I/O clustering is a technique of grouping multiple I/O operations together for improved performance. VxFS I/O policies provide more aggressive clustering processes than other file systems and offer higher I/O throughput when using large files. The resulting performance is comparable to that provided by raw disk.

## VxVM Integration

VxFS interfaces with VxVM to determine the I/O characteristics of the underlying volume and perform I/O accordingly. VxFS also uses this information when using `mkfs` to perform proper allocation unit alignments for efficient I/O operations from the kernel.

As part of VxFS/VxVM integration, VxVM exports a set of I/O parameters to achieve better I/O performance. This interface can enhance performance for different volume configurations such as RAID-5, striped, and mirrored volumes. Full stripe writes are important in a RAID-5 volume for strong I/O performance. VxFS uses these parameters to issue appropriate I/O requests to VxVM.

## Application-Specific Parameters

You can also set application specific parameters on a per-file system basis to improve I/O performance.

- Discovered Direct I/O

  All sizes above this value would be performed as direct I/O.

- Maximum Direct I/O Size

  This value defines the maximum size of a single direct I/O.

For a discussion on VxVM integration and performance benefits, refer to "VxFS Performance: Creating, Mounting, and Tuning File Systems" on page 21, "Application Interface" on page 53, and the `vxtunefs`(1M) and `tunefstab`(1M) manual pages.



# Access Control Lists

An Access Control List (ACL) stores a series of entries that identify specific users or groups and their access privileges for a directory or file. A file may have its own ACL or may share an ACL with other files. ACLs have the advantage of specifying detailed access permissions for multiple users and groups. Refer to the `getacl`(1) and `setacl`(1) manual pages for information on viewing and setting ACLs. For VxFS file systems created with the Version 5 disk layout, up to 1024 ACL entries can be specified. ACLs are also supported on cluster file systems.

# Storage Checkpoints

To increase availability, recoverability, and performance, the VERITAS File System offers on-disk and online backup and restore capabilities that facilitate frequent and efficient backup strategies. Backup and restore applications can leverage the VERITAS *Storage Checkpoint*, a disk- and I/O-efficient copying technology for creating periodic *frozen images* of a file system. Storage Checkpoints present a view of a file system at a point in time, and subsequently identifies and maintains copies of the original file system blocks. Instead of using a disk-based mirroring method, Storage Checkpoints save disk space and significantly reduce I/O overhead by using the free space pool available to a file system.

See "Storage Checkpoints" on page 65 for information on using the Storage Checkpoint feature. Storage Checkpoint functionality is a separately licensable feature.

# Online Backup

VxFS provides online data backup using the *snapshot* feature. An image of a mounted file system instantly becomes an exact read-only copy of the file system at a specific point in time. The original file system is called the *snapped* file system, the copy is called the *snapshot*.

When changes are made to the snapped file system, the old data is copied to the snapshot. When the snapshot is read, data that has not changed is read from the snapped file system, changed data is read from the snapshot.

Backups require one of the following methods:

- Copying selected files from the snapshot file system (using `find` and `cpio`)
- Backing up the entire file system (using `fscat`)
- Initiating a full or incremental backup (using `vxdump`)

See "Online Backup Using File System Snapshots" on page 89 for information on doing backups using the snapshot feature.

# Quotas

VxFS supports quotas, which allocate per-user quotas and limit the use of two principal resources: files and data blocks. You can assign quotas for each of these resources. Each quota consists of two limits for each resource, the *hard limit* and *soft limit*.

The *hard limit* represents an absolute limit on data blocks or files. A user can never exceed the hard limit under any circumstances.

The *soft limit* is lower than the hard limit and can be exceeded for a limited amount of time. This allows users to exceed limits temporarily as long as they fall under those limits before the allotted time expires.

See "Quota Limits" on page 98 for details on using VxFS quotas.

# Support for Databases

Databases are usually created on file systems to simplify backup, copying, and moving tasks and are slower compared to databases on raw disks.

Using the VERITAS Quick I/O for Databases feature with VxFS lets systems retain the benefits of having a database on a file system without sacrificing performance. VERITAS Quick I/O creates regular, preallocated files to use as character devices. Databases can be created on the character devices to achieve the same performance as databases created on raw disks.

Treating regular VxFS files as raw devices has the following advantages for databases:

- Commercial database servers such as OracleServer can issue kernel supported asynchronous I/O calls (through the asyncdsk or Posix AIO interface) on these pseudo devices but not on regular files.

# Cluster File Systems

Clustered file systems are an extension of VxFS that support concurrent direct media access from multiple systems. CFS employs a master/slave protocol. All cluster file systems can read file data directly from a shared disk. In addition, all systems can write "in-place" file data. Operations that require changes to file system metadata, such as allocation, creation, and deletion, can only be performed by the single primary file system node. To maintain file system consistency, secondary nodes must send messages to the primary, and the primary will perform the operations.



Installing VxFS and enabling the cluster feature does not create a cluster file system configuration. File system clustering requires other VERITAS products to enable communication services and provide storage resources. These products are packaged with VxFS in the Storage Foundation Cluster File System to provide a complete clustering environment.

See the *VERITAS SANPoint Foundation Suite Installation and Configuration Guide*, included in the VERITAS SANPoint Foundation Suite product, for more information.

To be a *cluster mount*, a file system must be mounted using the `mount -o cluster` option. File systems mounted without the `-o cluster` option are termed *local mounts.*

CFS functionality is a separately licensable feature.

# Cross-Platform Data Sharing

Cross-platform data sharing allows data to be *serially* shared among heterogeneous systems where each system has direct access to the physical devices that hold the data. This feature can be used only in conjunction with VERITAS Volume Manager. See the *VERITAS Storage Foundation Cross-Platform Data Sharing Administrator's Guide* for more information.

# File Change Log

The VxFS File Change Log (FCL) tracks changes to files and directories in a file system. The File Change Log can be used by applications such as backup products, webcrawlers, search and indexing engines, and replication software that typically scan an entire file system searching for modifications since a previous scan. FCL functionality is a separately licensable feature. See "File Change Log" on page 101 for more information.

# Multi-Volume Support

The multi-volume support (MVS) feature allows several volumes to be represented by a single logical object. All I/O to and from an underlying logical volume is directed by way of *volume sets.* This feature can be used only in conjunction with VERITAS Volume Manager. MVS functionality is a separately licensable feature. See "Multi-Volume File Systems" on page 107 for more information.

# Quality of Storage Service

The Quality of Storage Service (QoSS) option is built on the multi-volume support technology introduced in this release. Using QoSS, you can map more than one volume to a single file system. You can then configure policies that automatically relocate files from one volume to another, or relocate files by running file relocation commands. Having multiple volumes lets you determine where files are located, which can improve performance for applications that access specific types of files. QoSS functionality is a separately licensable feature that is available with the `VRTSfppm` package. See "Quality of Storage Service" on page 115 for more information.



# VxFS Performance: Creating, Mounting, and Tuning File Systems 2

For any file system, the ability to provide peak performance is important. Adjusting the available VERITAS File System (VxFS) options provides a way to increase system performance. This chapter describes the commands and practices you can use to optimize VxFS. For information on optimizing an application for use with VxFS, see "Application Interface" on page 53.

The following topics are covered in this chapter:





# Choosing mkfs Command Options

There are several characteristics that you can select when you create a file system. The most important options pertaining to system performance are the block size and intent log size.

## Block Size

The unit of allocation in VxFS is a block. Unlike some other UNIX file systems, VxFS does not make use of block *fragments* for allocation because storage is allocated in extents that consist of one or more blocks.

You specify the block size when creating a file system by using the `mkfs -o bsize` option. The block size cannot be altered after the file system is created. The smallest available block size for VxFS is 1K, which is also the default block size.

Choose a block size based on the type of application being run. For example, if there are many small files, a 1K block size may save space. For large file systems, with relatively few files, a larger block size is more appropriate. Larger block sizes use less disk space in file system overhead, but consume more space for files that are not a multiple of the block size. The easiest way to judge which block sizes provide the greatest system efficiency is to try representative system loads against various sizes and pick the fastest.

## Intent Log Size

You specify the intent log size when creating a file system by using the `mkfs -o logsize` option. With the Version 6 disk layout, you can dynamically increase or decrease the intent log size using the `log` option of the `fsadm` command. The `mkfs` utility uses a default intent log size of 16 megabytes for disk layout Versions 4, 5, and 6. The default size is sufficient for most workloads. If the system is used as an NFS server or for intensive synchronous write workloads, performance may be improved using a larger log size.

With larger intent log sizes, recovery time is proportionately longer and the file system may consume more system resources (such as memory) during normal operation.

There are several system performance benchmark suites for which VxFS performs better with larger log sizes. As with block sizes, the best way to pick the log size is to try representative system loads against various sizes and pick the fastest.

# Choosing mount Command Options

In addition to the standard mount mode (`delaylog` mode), VxFS provides `blkclear`, `log`, `tmplog`, and `nodatainlog` modes of operation. Caching behavior can be altered with the `mincache` option, and the behavior of `O_SYNC` and `D_SYNC` (see the `fcntl`(2) manual page) writes can be altered with the `convosync` option.

The `delaylog` and `tmplog` modes can significantly improve performance. The improvement over `log` mode is typically about 15 to 20 percent with `delaylog`; with `tmplog`, the improvement is even higher. Performance improvement varies, depending on the operations being performed and the workload. Read/write intensive loads should show less improvement, while file system structure intensive loads (such as `mkdir`, `create`, and `rename`) may show over 100 percent improvement. The best way to select a mode is to test representative system loads against the logging modes and compare the performance results.

Most of the modes can be used in combination. For example, a desktop machine might use both the `blkclear` and `mincache=closesync` modes.

Additional information on `mount` options can be found in the `mount_vxfs`(1M) manual page.

In the following descriptions, the term “effects of system calls” refers to changes to file system data and metadata caused by the system call, excluding changes to st_atime (see the `stat`(2) manual page).

## log

In `log` mode, all system calls other than `write`(2), `writev`(2), and `pwrite`(2) are guaranteed to be persistent once the system call returns to the application.

The `rename`(2) system call flushes the source file to disk to guarantee the persistence of the file data before renaming it. In both modes, the rename is also guaranteed to be persistent when the system call returns. This benefits shell scripts and programs that try to update a file atomically by writing the new file contents to a temporary file and then renaming it on top of the target file.



## delaylog

The default logging mode is `delaylog`. In delaylog mode, the effects of most system calls other than `write`(2), `writev`(2), and `pwrite`(2) are guaranteed to be persistent approximately 3 seconds after the system call returns to the application. Contrast this with the behavior of most other file systems in which most system calls are not persistent until approximately 30 seconds or more after the call has returned. Fast file system recovery works with this mode.

The `rename`(2) system call flushes the source file to disk to guarantee the persistence of the file data before renaming it. In both modes, the rename is also guaranteed to be persistent when the system call returns. This benefits shell scripts and programs that try to update a file atomically by writing the new file contents to a temporary file and then renaming it on top of the target file.

## tmplog

In tmplog mode, the effects of system calls have persistence guarantees that are similar to those in delaylog mode. In addition, enhanced flushing of delayed extending writes is disabled, which results in better performance but increases the chances of data being lost or unitialized data appearing in a file that was being actively written at the time of a system failure. This mode is only recommended for temporary file systems. Fast file system recovery works with this mode.

---

**Note** In all logging modes, VxFS is fully POSIX compliant. The effects of the `fsync`(2) and `fdatasync`(2) system calls are guaranteed to be persistent once the calls return. The persistence guarantees for data or metadata modified by `write`(2), `writev`(2), or `pwrite`(2) are not affected by the logging mount options. The effects of these system calls are guaranteed to be persistent only if the `O_SYNC`, `O_DSYNC`, `VX_DSYNC`, or `VX_DIRECT` flag, as modified by the `convosync= mount` option, has been specified for the file descriptor.

The behavior of NFS servers on a VxFS file system is unaffected by the log and tmplog mount options, but not delaylog. In all cases except with delaylog, VxFS complies with the persistency requirements of the NFS v2 and NFS v3 standard.

Unless a UNIX application has been developed specifically for the VxFS file system in `log` mode, it will expect the persistence guarantees offered by most other file systems and will experience improved robustness when used with a VxFS file system mounted in `delaylog` mode. Applications that expect better persistence guarantees than that offered by most other file systems can benefit from the `log`, `mincache=`, and `closesync` `mount` options. However, most commercially available applications will work well with the default VxFS `mount` options, including the `delaylog` mode.

---

## logiosize

The `logiosize=size` option is provided to enhance the performance of storage devices that employ a *read-modify-write* feature. If you specify `logiosize` when you mount a file system, VxFS writes the intent log in at least `size` bytes to obtain the maximum performance from such devices. The values for `size` can be 1024, 2048, or 4096.

## nodatainlog

Use the `nodatainlog` mode on systems with disks that do not support bad block revectoring. Usually, a VxFS file system uses the intent log for synchronous writes. The inode update and the data are both logged in the transaction, so a synchronous write only requires one disk write instead of two. When the synchronous write returns to the application, the file system has told the application that the data is already written. If a disk error causes the metadata update to fail, then the file must be marked bad and the entire file is lost.

If a disk supports bad block revectoring, then a failure on the data update is unlikely, so logging synchronous writes should be allowed. If the disk does not support bad block revectoring, then a failure is more likely, so the `nodatainlog` mode should be used.

A `nodatainlog` mode file system is approximately 50 percent slower than a standard mode VxFS file system for synchronous writes. Other operations are not affected.

## blkclear

The `blkclear` mode is used in increased data security environments. The `blkclear` mode guarantees that uninitialized storage never appears in files. The increased integrity is provided by clearing extents on disk when they are allocated within a file. Extending writes are not affected by this mode. A `blkclear` mode file system is approximately 10 percent slower than a standard mode VxFS file system, depending on the workload.

## mincache

The `mincache` mode has five suboptions:

- `mincache=closesync`
- `mincache=direct`
- `mincache=dsync`
- `mincache=unbuffered`
- `mincache=tmpcache`



The `mincache=closesync` mode is useful in desktop environments where users are likely to shut off the power on the machine without halting it first. In this mode, any changes to the file are flushed to disk when the file is closed.

To improve performance, most file systems do not synchronously update data and inode changes to disk. If the system crashes, files that have been updated within the past minute are in danger of losing data. With the `mincache=closesync` mode, if the system crashes or is switched off, only files that are currently open can lose data. A `mincache=closesync` mode file system should be approximately 15 percent slower than a standard mode VxFS file system, depending on the workload.

The `mincache=direct`, `mincache=unbuffered`, and `mincache=dsync` modes are used in environments where applications are experiencing reliability problems caused by the kernel buffering of I/O and delayed flushing of non-synchronous I/O. The `mincache=direct` and `mincache=unbuffered` modes guarantee that all non-synchronous I/O requests to files will be handled as if the `VX_DIRECT` or `VX_UNBUFFERED` caching advisories had been specified. The `mincache=dsync` mode guarantees that all non-synchronous I/O requests to files will be handled as if the `VX_DSYNC` caching advisory had been specified. Refer to the `vxfsio`(7) manual page for explanations of `VX_DIRECT`, `VX_UNBUFFERED`, and `VX_DSYNC`, as well as for the requirements for direct I/O. The `mincache=direct`, `mincache=unbuffered`, and `mincache=dsync` modes also flush file data on `close` as `mincache=closesync` does.

Because the `mincache=direct`, `mincache=unbuffered`, and `mincache=dsync` modes change non-synchronous I/O to synchronous I/O, there can be a substantial degradation in throughput for small to medium size files for most applications. Since the `VX_DIRECT` and `VX_UNBUFFERED` advisories do not allow any caching of data, applications that would normally benefit from caching for reads will usually experience less degradation with the `mincache=dsync` mode. `mincache=direct` and `mincache=unbuffered` require significantly less CPU time than buffered I/O.

If performance is more important than data integrity, you can use the `mincache=tmpcache` mode. The `mincache=tmpcache` mode disables special delayed extending write handling, trading off less integrity for better performance. Unlike the other `mincache` modes, `tmpcache` does not flush the file to disk when it is closed. When the `mincache=tmpcache` option is used, bad data can appear in a file that was being extended when a crash occurred.

## convosync

**Note** Use of the `convosync=dsync` option violates POSIX guarantees for synchronous I/O.

The `convosync` (convert osync) mode has five suboptions:

- `convosync=closesync`
- `convosync=delay`
- `convosync=direct`
- `convosync=dsync`
- `convosync=unbuffered`

The `convosync=closesync` mode converts synchronous and data synchronous writes to non-synchronous writes and flushes the changes to the file to disk when the file is closed.



The convosync=delay mode causes synchronous and data synchronous writes to be delayed rather than to take effect immediately. No special action is performed when closing a file. This option effectively cancels any data integrity guarantees normally provided by opening a file with O_SYNC. See the open(2), fcntl(2), and vxfsio(7) manual pages for more information on O_SYNC.

**Caution** Be very careful when using the convosync=closesync or convosync=delay mode because they actually change synchronous I/O into non-synchronous I/O. This may cause applications that use synchronous I/O for data reliability to fail if the system crashes and synchronously written data is lost.

The convosync=direct and convosync=unbuffered mode convert synchronous and data synchronous reads and writes to direct reads and writes.

The convosync=dsync mode converts synchronous writes to data synchronous writes.

As with closesync, the direct, unbuffered, and dsync modes flush changes to the file to disk when it is closed. These modes can be used to speed up applications that use synchronous I/O. Many applications that are concerned with data integrity specify the O_SYNC fcntl in order to write the file data synchronously. However, this has the undesirable side effect of updating inode times and therefore slowing down performance. The convosync=dsync, convosync=unbuffered, and convosync=direct modes alleviate this problem by allowing applications to take advantage of synchronous writes without modifying inode times as well.

**Caution** Before using convosync=dsync, convosync=unbuffered, or convosync=direct, make sure that all applications that use the file system do not require synchronous inode time updates for O_SYNC writes.

## ioerror

Sets the policy for handling I/O errors on a mounted file system. I/O errors can occur while reading or writing file data, or while reading or writing metadata. The file system can respond to these I/O errors either by halting or by gradually degrading. The ioerror option provides four policies that determine how the file system responds to the various errors. All four policies limit data corruption, either by stopping the file system or by marking a corrupted inode as bad. The four policies are disable, nodisable, wdisable, and mwdisable.

If `disable` is selected, VxFS disables the file system after detecting any I/O error. You must then unmount the file system and correct the condition causing the I/O error. After the problem is repaired, run `fsck` and mount the file system again. In most cases, replaying `fsck` is sufficient to repair the file system. A full `fsck` is required only in cases of structural damage to the file system's metadata. Select `disable` in environments where the underlying storage is redundant, such as RAID-5 or mirrored disks.

If `nodisable` is selected, when VxFS detects an I/O error, it sets the appropriate error flags to contain the error, but continues running. Note that the "degraded" condition indicates possible data or metadata corruption, not the overall performance of the file system.

For file data read and write errors, VxFS sets the VX_DATAIOERR flag in the super-block. For metadata read errors, VxFS sets the VX_FULLFSCK flag in the super-block. For metadata write errors, VxFS sets the VX_FULLFSCK and VX_METAIOERR flags in the super-block and may mark associated metadata as bad on disk. VxFS then prints the appropriate error messages to the console (see "Kernel Messages" on page 163 for information on actions to take for specific errors).

You should stop the file system as soon as possible and repair the condition causing the I/O error. After the problem is repaired, run `fsck` and mount the file system again. Select `nodisable` if you want to implement the policy that most closely resembles the error handling policy of the previous VxFS release.

If `wdisable` (write disable) or `mwdisable` (metadata-write disable) is selected, the file system is disabled or degraded, depending on the type of error encountered. Select `wdisable` or `mwdisable` for environments where read errors are more likely to persist than write errors, such as when using non-redundant storage. `mwdisable` is the default ioerror mount option for local mounts. See the `mount_vxfs`(1M) manual page for more information.

## largefiles | nolargefiles

VxFS supports files larger than 2 gigabytes. The maximum file size that can be created is 2 terabytes.

**Note** Applications and utilities such as backup may experience problems if they are not aware of large files. In such a case, create your file system without large file capability.



## Creating a File System with Large Files

You can create a file system with large file capability by entering the following command:

```
# mkfs -F vxfs -o largefiles special_device size
```

Specifying `largefiles` sets the `largefiles` flag, which allows the file system to hold files that are two terabytes or larger in size. The default option is `largefiles`. Conversely, the `nolargefiles` option clears the flag and prevents large files from being created:

```
# mkfs -F vxfs -o nolargefiles special_device size
```

**Note** The `largefiles` flag is persistent and stored on disk.

## Mounting a File System with Large Files

If a mount succeeds and `nolargefiles` is specified, the file system cannot contain or create any large files. If a mount succeeds and `largefiles` is specified, the file system may contain and create large files.

The `mount` command fails if the specified `largefiles|nolargefiles` option does not match the on-disk flag.

The `mount` command defaults to match the current setting of the on-disk flag if specified without the `largefiles` or `nolargefiles` option, so it's best not to specify either option. After a file system is mounted, you can use the `fsadm` utility to change the large files option.

## Managing a File System with Large Files

You can determine the current status of the largefiles flag using the `fsadm` or `mkfs` command:

```
# mkfs -F vxfs -m special_device
# fsadm -F vxfs mount_point | special_device
```

You can switch capabilities on a mounted file system using the `fsadm` command:

```
# fsadm -F vxfs -o [no]largefiles mount_point
```

You can also switch capabilities on an unmounted file system:

```
# fsadm -F vxfs -o [no]largefiles special_device
```

You cannot change a file system to `nolargefiles` if it holds large files.

See the `mount_vxfs`(1M), `fsadm_vxfs`(1M), and `mkfs_vxfs`(1M) manual pages.

## Combining mount Command Options

Although `mount` options can be combined arbitrarily, some combinations do not make sense. The following examples provide some common and reasonable `mount` option combinations.

### Example 1 - Desktop File System

```
# mount -F vxfs -o log,mincache=closesync /dev/dsk/c1t3d0 /mnt
```

This guarantees that when a file is closed, its data is synchronized to disk and cannot be lost. Thus, once an application is exited and its files are closed, no data will be lost even if the system is immediately turned off.

### Example 2 - Temporary File System or Restoring from Backup

```
# mount -F vxfs -o tmplog,convosync=delay,mincache=tmpcache \
  /dev/dsk/c1t3d0 /mnt
```

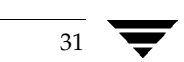
This combination might be used for a temporary file system where performance is more important than absolute data integrity. Any `O_SYNC` writes are performed as delayed writes and delayed extending writes are not handled specially (which could result in a file that contains garbage if the system crashes at the wrong time). Any file written 30 seconds or so before a crash may contain garbage or be missing if this `mount` combination is in effect. However, such a file system will do significantly less disk writes than a `log` file system, and should have significantly better performance, depending on the application.

### Example 3 - Data Synchronous Writes

```
# mount -F vxfs -o log,convosync=dsync /dev/dsk/c1t3d0 /mnt
```

This combination would be used to improve the performance of applications that perform `O_SYNC` writes, but only require data synchronous write semantics. Their performance can be significantly improved if the file system is mounted using `convosync=dsync` without any loss of data integrity.





# Kernel Tunables

This section describes the kernel tunable parameters in VxFS.

## Internal Inode Table Size

VxFS caches inodes in an *inode table*. There is a dynamic tunable in VxFS called `vx_ninode` that determines the number of entries in the inode table.

You can dynamically change the value of `vx_ninode` by using the `sam` or `kctune` commands (see the `sam`(1M) and `kctune`(1M) manual pages). This value is used to determine the number of entries in the VxFS inode table. By default, `vx_ninode` initializes at zero; the file system then computes a value based on the system memory size.

A VxFS file system can also obtain the value of `vx_ninode` from the system configuration file used for making the HP-UX kernel (`/stand/system` for example). To change the computed value of `vx_ninode`, you can add an entry to the system configuration file. For example:

```
tunable vx_ninode 1000000
```

This sets the inode table size to 1,000,000 inodes after making a new HP-UX kernel using `mk_kernel`.

Increasing the value of `vx_ninode` increases the inode table size immediately, allowing a higher number of inodes to be cached. Decreasing the value of `vx_ninode` decreases the inode table size to the specified value. After the tunable is decreased, VxFS attempts to free excess cached objects so that the resulting number of inodes in the table is less than or equal to the specified value of `vx_ninode`. If this attempt fails, the value of the `vx_ninode` tunable is not changed. In such a case, the `kctune` command can be specified with the `-h` option so that the new value of `vx_ninode` takes effect after a system reboot.

Be careful when changing the value of `vx_ninode`, as the value can affect file system performance. Typically, the default value determined by VxFS based on the amount of system memory ensures good system performance across a wide range of applications. However, if it is determined that the default value is not suitable, `vx_ninode` can be set to an appropriate value based on the expected file system usage. The `vxfsstat` command can be used to monitor inode cache usage and statistics to determine the optimum value of `vx_ninode` for the system.

Changing the value of a tunable does not resize the internal hash tables and structures of the caches. These sizes are determined at system boot up based on either the system memory size, which is the default, or the value of the tunable if explicitly set, whichever is larger. Thus, dynamically increasing the tunable to a value that is more than two times either the default value or the user-defined value, if larger, may cause performance degradation unless the system is rebooted.

## Examples of Changing the vx_inode Tunable Value

The following are examples of changing the vx_ninode tunable value.

### Example 1 - Reporting the Current Value of vx_ninode

```
# kctune vx_ninode
```

This command displays the current value of vx_ninode.

### Example 2 - Setting vx_ninode

```
# kctune -s vx_ninode=10000
```

This command sets vx_ninode to 10000, the specified value.

### Example 3 - Restoring vx_ninode to Its Default Value

```
# kctune -s vx_ninode=
```

This command restores vx_ninode to its default value by clearing the user-specified value. The default value is the value determined by VxFS to be optimal based on the amount of system memory, which is used if vx_ninode is not explicitly set.

### Example 4 - Delaying a Change to vx_ninode Until After a Reboot

```
# kctune -h -s vx_ninode=10000
```

If the -h option is specified, the specified value for vx_ninode will not take effect until after a system reboot.

## VxFS Buffer Cache High Water Mark

VxFS maintains its own buffer cache in the kernel for frequently accessed file system metadata. This cache is different from the HP-UX kernel buffer cache that caches file data. The vx_bc_bufhwm dynamic, global, tunable parameter lets you change the VxFS buffer cache *high water mark*, that is, the maximum amount of memory that can be used to cache VxFS metadata.

The initial value of vx_bc_bufhwm is zero. When the operating system reboots, VxFS sets the value of vx_bc_bufhwm based on the amount of system memory. You can explicitly reset the value of vx_bc_bufhwm by changing the value of vxfs_bc_bufhwm using the sam or kctune commands (see the sam(1M) and kctune(1M) manual pages). You can also set the value by adding an entry to the system configuration file. For example, the following entry:



```
vxfs_bc_bufhwm vx_bc_bufhwm  300000
```

sets the high water mark to 300 megabytes. The change takes effect after you rebuild the HP-UX kernel using the `mk_kernel` command. You specify the `vx_bc_bufhwm` tunable in units of kilobytes. The minimum value is 6144.

Increasing the value of `vx_bc_bufhwm` increases the VxFS buffer cache immediately, allowing a greater amount of memory to be used to cache VxFS metadata. Decreasing the value of `vx_bc_bufhwm` decreases the VxFS buffer cache to the specified value. This frees memory such that the amount of memory used for buffer cache is lower than the specified value of `vx_bc_bufhwm`.

Typically, the default value computed by VxFS based on the amount of system memory ensures good system performance across a wide range of applications. For application loads that cause frequent file system metadata changes on the system (for example, a high rate of file creation or deletion, or accessing large directories), changing the value of `vx_bc_bufhwm` may improve performance.

You can use the `vxfsstat` command to monitor buffer cache statistics and inode cache usage (see the `vxfsstat`(1M) manual page).

## Number of Links to a File

In VxFS, the number of possible links to a file is determined by the `vx_maxlink` global tunable. The default value of `vx_maxlink` is 32767, the maximum value is 65535. This is a static tunable.

You can set the value of `vx_maxlink` using the `sam` or `kctune` commands (see the `sam`(1M) and `kctune`(1M) manual pages), or by adding an entry to the system configuration file as shown in the following example:

```
vxfs_maxlink vx_maxlink 40000
```

This sets the value of `vx_maxlink` to 40,000 links.

## VxFS Inode Free Time Lag

In VxFS, an inode is put on a *freelist* if it is not being used. The memory space for this unused inode can be freed if it stays on the *freelist* for a specified amount of time. The `vx_ifree_timelag` tunable specifies the minimum amount of time an unused inode spends on a *freelist* before its memory space is freed.

The `vx_ifree_timelag` tunable is dynamic. Any changes to `vx_ifree_timelag` take effect immediately.

The default value of `vx_ifree_timelag` is 0. By setting `vx_ifree_timelag` to 0, the inode free time lag is autotuned to 1800 seconds. Specifying negative one (-1) stops the freeing of inode space; no further inode allocations are freed until the value is changed back to a value other than negative one.

You can change the value of `vx_ifree_timelag` using the `sam` or `kctune` commands (see the `sam`(1M) and `kctune`(1M) manual pages), or by adding an entry to the system configuration file. The following example changes the value of `vx_ifree_timelag` to 2400 seconds:

```
# kctune -s vxfs_ifree_timelag=2400
```

**Note** The default value `vx_ifree_timelag` typically provides optimal VxFS performance. Be careful when adjusting the tunable because incorrect tuning can adversely affect system performance.

### VxVM Maximum I/O Size

When using VxFS with the VERITAS Volume Manager (VxVM), VxVM by default breaks up I/O requests larger than 256K. When using striping, to optimize performance, the file system issues I/O requests that are up to a full stripe in size. If the stripe size is larger than 256K, those requests are broken up.

See the *VERITAS Volume Manager Administrator's Guide* for more information on avoiding I/O breakup by setting the maximum I/O tunable parameter.

## Monitoring Free Space

In general, VxFS works best if the percentage of free space in the file system does not get below 10 percent. This is because file systems with 10 percent or more free space have less fragmentation and better extent allocation. Regular use of the `df` command (see the `df_vxfs`(1M) manual page) to monitor free space is desirable. Full file systems may have an adverse effect on file system performance. Full file systems should therefore have some files removed, or should be expanded (see the `fsadm_vxfs`(1M) manual page for a description of online file system expansion).



## Monitoring Fragmentation

Fragmentation reduces performance and availability. Regular use of `fsadm`'s fragmentation reporting and reorganization facilities is therefore advisable.

The easiest way to ensure that fragmentation does not become a problem is to schedule regular defragmentation runs using the `cron` command.

Defragmentation scheduling should range from weekly (for frequently used file systems) to monthly (for infrequently used file systems). Extent fragmentation should be monitored with `fsadm` or the `df -o s` commands. There are three factors which can be used to determine the degree of fragmentation:

- Percentage of free space in extents of less than 8 blocks in length
- Percentage of free space in extents of less than 64 blocks in length
- Percentage of free space in extents of length 64 blocks or greater

An unfragmented file system will have the following characteristics:

- Less than 1 percent of free space in extents of less than 8 blocks in length
- Less than 5 percent of free space in extents of less than 64 blocks in length
- More than 5 percent of the total file system size available as free extents in lengths of 64 or more blocks

A badly fragmented file system will have one or more of the following characteristics:

- Greater than 5 percent of free space in extents of less than 8 blocks in length
- More than 50 percent of free space in extents of less than 64 blocks in length
- Less than 5 percent of the total file system size available as free extents in lengths of 64 or more blocks

The optimal period for scheduling of extent reorganization runs can be determined by choosing a reasonable interval, scheduling `fsadm` runs at the initial interval, and running the extent fragmentation report feature of `fsadm` before and after the reorganization.

The "before" result is the degree of fragmentation prior to the reorganization. If the degree of fragmentation is approaching the figures for bad fragmentation, reduce the interval between `fsadm` runs. If the degree of fragmentation is low, increase the interval between `fsadm` runs.

The "after" result is an indication of how well the reorganizer has performed. The degree of fragmentation should be close to the characteristics of an unfragmented file system. If not, it may be a good idea to resize the file system; full file systems tend to fragment and are difficult to defragment. It is also possible that the reorganization is not being performed at a time during which the file system in question is relatively idle.

Directory reorganization is not nearly as critical as extent reorganization, but regular directory reorganization will improve performance. It is advisable to schedule directory reorganization for file systems when the extent reorganization is scheduled. The following is a sample script that is run periodically at 3:00 A.M. from `cron` for a number of file systems:

```
outfile=/usr/spool/fsadm/out.‘/bin/date +’%m%d’‘
  for i in /home /home2 /project /db
  do
    /bin/echo "Reorganizing $i"
    /bin/timex fsadm -F vxfs -e -E -s $i
    /bin/timex fsadm -F vxfs -s -d -D $i
  done > $outfile 2>&1
```

# I/O Tuning

**Note** The tunables and the techniques described in this section work on a per file system basis. Use them judiciously based on the underlying device properties and characteristics of the applications that use the file system.

Performance of a file system can be enhanced by a suitable choice of I/O sizes and proper alignment of the I/O requests based on the requirements of the underlying special device. VxFS provides tools to tune the file systems.

## Tuning VxFS I/O Parameters

VxFS provides a set of tunable I/O parameters that control some of its behavior. These I/O parameters are useful to help the file system adjust to striped or RAID-5 volumes that could yield performance superior to a single disk. Typically, data streaming applications that access large files see the largest benefit from tuning the file system.

The file system queries VxVM to determine the geometry of the underlying volume and automatically sets the I/O parameters. The `mount` command also queries VxVM when the file system is mounted and downloads the I/O parameters.

If the default parameters are not acceptable or the file system is being used without VxVM, then the `/etc/vx/tunefstab` file can be used to set values for I/O parameters. The `mount` command reads the `/etc/vx/tunefstab` file and downloads any parameters specified for a file system. The `tunefstab` file overrides any values obtained from VxVM. While the file system is mounted, any I/O parameters can be changed using the `vxtunefs` command which can have tunables specified on the command line or can read them from the `/etc/vx/tunefstab` file. For more details, see the `vxtunefs`(1M) and `tunefstab`(4) manual pages. The `vxtunefs` command can be used to print the current values of the I/O parameters:



```
# vxtunefs -p mount_point
```

The following is an example `tunefstab` file:

```
/dev/vx/dsk/userdg/netbackup
read_pref_io=128k,write_pref_io=128k,read_nstream=4,write_nstream=4
/dev/vx/dsk/userdg/metasave
read_pref_io=128k,write_pref_io=128k,read_nstream=4,write_nstream=4
/dev/vx/dsk/userdg/solbuild
read_pref_io=64k,write_pref_io=64k,read_nstream=4,write_nstream=4
/dev/vx/dsk/userdg/solrelease
read_pref_io=64k,write_pref_io=64k,read_nstream=4,write_nstream=4
/dev/vx/dsk/userdg/solpatch
read_pref_io=128k,write_pref_io=128k,read_nstream=4,write_nstream=4
```

## Tunable VxFS I/O Parameters

| | |
|---|---|
| `read_pref_io` | The preferred read request size. The file system uses this in conjunction with the `read_nstream` value to determine how much data to read ahead. The default value is 64K. |
| `write_pref_io` | The preferred write request size. The file system uses this in conjunction with the `write_nstream` value to determine how to do flush behind on writes. The default value is 64K. |
| `read_nstream` | The number of parallel read requests of size `read_pref_io` to have outstanding at one time. The file system uses the product of `read_nstream` multiplied by `read_pref_io` to determine its read ahead size. The default value for `read_nstream` is 1. |
| `write_nstream` | The number of parallel write requests of size `write_pref_io` to have outstanding at one time. The file system uses the product of `write_nstream` multiplied by `write_pref_io` to determine when to do flush behind on writes. The default value for `write_nstream` is 1. |
| `discovered_direct_iosz` | Any file I/O requests larger than the `discovered_direct_iosz` are handled as discovered direct I/O. A discovered direct I/O is unbuffered similar to direct I/O, but it does not require a synchronous commit of the inode when the file is extended or blocks are allocated. For larger I/O requests, the CPU time for copying the data into the page cache and the cost of using memory to buffer the I/O data becomes more expensive than the cost of doing the disk I/O. For these I/O requests, using discovered direct I/O is more efficient than regular I/O. The default value of this parameter is 256K. |



`default_indir_size`

On VxFS, files can have up to ten direct extents of variable size stored in the inode. Once these extents are used up, the file must use indirect extents which are a fixed size that is set when the file first uses indirect extents. These indirect extents are 8K by default. The file system does not use larger indirect extents because it must fail a write and return `ENOSPC` if there are no extents available that are the indirect extent size. For file systems with many large files, the 8K indirect extent size is too small. The files that get into indirect extents use many smaller extents instead of a few larger ones. By using this parameter, the default indirect extent size can be increased so large that files in indirects use fewer larger extents. The tunable `default_indir_size` should be used carefully. If it is set too large, then writes will fail when they are unable to allocate extents of the indirect extent size to a file. In general, the fewer and the larger the files on a file system, the larger the `default_indir_size` can be set. This parameter should generally be set to some multiple of the `read_pref_io` parameter. `default_indir_size` is not applicable on Version 4 disk layouts.

`fcl_keeptime`

Specifies the minimum amount of time, in seconds, that the VxFS file change log (FCL) keeps records in the log. When the oldest 8K block of FCL records have been kept longer than the value of `fcl_keeptime`, they are purged from the FCL and the extents nearest to the beginning of the FCL file are freed. This process is referred to as "punching a hole." Holes are punched in the FCL file in 8K chunks.

If the `fcl_maxalloc` parameter is set, records are purged from the FCL if the amount of space allocated to the FCL exceeds `fcl_maxalloc`, even if the elapsed time the records have been in the log is less than the value of `fcl_keeptime`. If the file system runs out of space before `fcl_keeptime` is reached, the FCL is deactivated.

Either or both of the `fcl_keeptime` or `fcl_maxalloc` parameters must be set before the file change log can be activated. `fcl_keeptime` does not apply to disk layout Versions 1 through 5.

`fcl_maxalloc`

Specifies the maximum amount of space that can be allocated to the VxFS file change log (FCL). The FCL file is a sparse file that grows as changes occur in the file system. When the space allocated to the FCL file reaches the `fcl_maxalloc` value, the oldest FCL records are purged from the FCL and the extents nearest to the beginning of the FCL file are freed. This process is referred to as "punching a hole." Holes are punched in the FCL file in 8K chunks. If the file system runs out of space before `fcl_maxalloc` is reached, the FCL is deactivated.

Either or both of the `fcl_maxalloc` or `fcl_keeptime` parameters must be set before the file change log can be activated. `fcl_maxalloc` does not apply to disk lay out Versions 1 through 5.

`fcl_winterval`

Specifies the time, in seconds, that must elapse before the VxFS file change log (FCL) records a data overwrite, data extending write, or data truncate for a file. The ability to limit the number of repetitive FCL records for continuous writes to the same file is important for file system performance and for applications processing the FCL. `fcl_winterval` is best set to an interval less than the shortest interval between reads of the FCL by any application. This way all applications using the FCL can be assured of finding at least one FCL record for any file experiencing continuous data changes.

`fcl_winterval` is enforced for all files in the file system. Each file maintains its own time stamps, and the elapsed time between FCL records is per file. This elapsed time can be overridden using the VxFS FCL sync public API (see the `vxfs_fcl_sync`(3) manual page). `fcl_winterval` does not apply to disk layout Versions 1 through 5.



`hsm_write_prealloc`

For a file managed by a hierarchical storage management (HSM) application, `hsm_write_prealloc` preallocates disk blocks before data is migrated back into the file system. An HSM application usually migrates the data back through a series of writes to the file, each of which allocates a few blocks. By setting `hsm_write_prealloc` (`hsm_write_prealloc`=1), a sufficient number of disk blocks are allocated on the first write to the empty file so that no disk block allocation is required for subsequent writes. This improves the write performance during migration.

The `hsm_write_prealloc` parameter is implemented outside of the DMAPI specification, and its usage has limitations depending on how the space within an HSM-controlled file is managed. It is advisable to use `hsm_write_prealloc` only when recommended by the HSM application controlling the file system.

`initial_extent_size`

Changes the default initial extent size. VxFS determines, based on the first write to a new file, the size of the first extent to be allocated to the file. Normally the first extent is the smallest power of 2 that is larger than the size of the first write. If that power of 2 is less than 8K, the first extent allocated is 8K. After the initial extent, the file system increases the size of subsequent extents (see `max_seqio_extent_size`) with each allocation. Since most applications write to files using a buffer size of 8K or less, the increasing extents start doubling from a small initial extent. `initial_extent_size` can change the default initial extent size to be larger, so the doubling policy will start from a much larger initial size and the file system will not allocate a set of small extents at the start of file. Use this parameter only on file systems that will have a very large average file size. On these file systems it will result in fewer extents per file and less fragmentation. `initial_extent_size` is measured in file system blocks.

`max_buf_data_size`

The maximum buffer size allocated for file data; either 8K bytes or 64K bytes. Use the larger value for workloads where large reads/writes are performed sequentially. Use the smaller value on workloads where the I/O is random or is done in small chunks. 8K bytes is the default value.

| | |
|---|---|
| `inode_aging_count` | Specifies the maximum number of inodes to place on an inode aging list. Inode aging is used in conjunction with file system Storage Checkpoints to allow quick restoration of large, recently deleted files. The aging list is maintained in first-in-first-out (fifo) order up to maximum number of inodes specified by `inode_aging_count`. As newer inodes are placed on the list, older inodes are removed to complete their aging process. For best performance, it is advisable to age only a limited number of larger files before completion of the removal process. The default maximum number of inodes to age is 2048. |
| `inode_aging_size` | Specifies the minimum size to qualify a deleted inode for inode aging. Inode aging is used in conjunction with file system Storage Checkpoints to allow quick restoration of large, recently deleted files. For best performance, it is advisable to age only a limited number of larger files before completion of the removal process. Setting the size too low can push larger file inodes out of the aging queue to make room for newly removed smaller file inodes. |
| `max_direct_iosz` | The maximum size of a direct I/O request that will be issued by the file system. If a larger I/O request comes in, then it is broken up into `max_direct_iosz` chunks. This parameter defines how much memory an I/O request can lock at once, so it should not be set to more than 20 percent of memory. |
| `max_diskq` | Limits the maximum disk queue generated by a single file. When the file system is flushing data for a file and the number of buffers being flushed exceeds `max_diskq`, processes will block until the amount of data being flushed decreases. Although this doesn't limit the actual disk queue, it prevents flushing processes from making the system unresponsive. The default value is 1 MB. |
| `max_seqio_extent_size` | Increases or decreases the maximum size of an extent. When the file system is following its default allocation policy for sequential writes to a file, it allocates an initial extent which is large enough for the first write to the file. When additional extents are allocated, they are progressively larger (the algorithm tries to double the size of the file with each new extent) so each extent can hold several writes worth of data. This is done to reduce the total number of extents in anticipation of continued sequential writes. When the file stops being written, any unused space is freed for other files to use. Normally this allocation stops increasing the size of extents at 2048 blocks which prevents one file from holding too much unused space. `max_seqio_extent_size` is measured in file system blocks. |



`qio_cache_enable` Enables or disables caching on Quick I/O files. The default behavior is to disable caching. To enable caching, set `qio_cache_enable` to 1. On systems with large memories, the database cannot always use all of the memory as a cache. By enabling file system caching as a second level cache, performance may be improved. If the database is performing sequential scans of tables, the scans may run faster by enabling file system caching so the file system will perform aggressive read-ahead on the files.

`read_ahead` The default for all VxFS read operations is to perform sequential read ahead. You can specify the `read_ahead` cache advisory to implement the VxFS *enhanced read ahead functionality*. This allows read aheads to detect more elaborate patterns (such as increasing or decreasing read offsets or multithreaded file accesses) in addition to simple sequential reads. You can specify the following values for `read_ahead`:

0—Disables read ahead functionality

1—Retains traditional sequential read ahead behavior

2—Enables enhanced read ahead for all reads

The default is 1—VxFS detects only sequential patterns.

`read_ahead` detects patterns on a per-thread basis, up to a maximum determined by `vx_era_nthreads` parameter. The default number of threads is 5, but you can change the default value by setting the `vx_era_nthreads` parameter in the `/etc/system` configuration file.

`write_throttle`

The `write_throttle` parameter is useful in special situations where a computer system has a combination of a large amount of memory and slow storage devices. In this configuration, sync operations (such as `fsync()`) may take long enough to complete that a system appears to hang. This behavior occurs because the file system is creating *dirty buffers* (in-memory updates) faster than they can be asynchronously flushed to disk without slowing system performance.

Lowering the value of `write_throttle` limits the number of dirty buffers per file that a file system will generate before flushing the buffers to disk. After the number of dirty buffers for a file reaches the `write_throttle` threshold, the file system starts flushing buffers to disk even if free memory is still available.

The default value of `write_throttle` is zero, which puts no limit on the number of dirty buffers per file. If non-zero, VxFS limits the number of dirty buffers per file to `write_throttle` buffers.

The default value typically generates a large number of dirty buffers, but maintains fast user writes. Depending on the speed of the storage device, if you lower `write_throttle`, user write performance may suffer, but the number of dirty buffers is limited, so sync operations will complete much faster.

Because lowering `write_throttle` may in some cases delay write requests (for example, lowering `write_throttle` may increase the file disk queue to the `max_diskq` value, delaying user writes until the disk queue decreases), it is advisable not to change the value of `write_throttle` unless your system has a combination of large physical memory and slow storage devices.



If the file system is being used with VxVM, it is advisable to let the VxFS I/O parameters get set to default values based on the volume geometry.

If the file system is being used with a hardware disk array or volume manager other than VxVM, try to align the parameters to match the geometry of the logical disk. With striping or RAID-5, it is common to set `read_pref_io` to the stripe unit size and `read_nstream` to the number of columns in the stripe. For striped arrays, use the same values for `write_pref_io` and `write_nstream`, but for RAID-5 arrays, set `write_pref_io` to the full stripe size and `write_nstream` to 1.

For an application to do efficient disk I/O, it should issue read requests that are equal to the product of `read_nstream` multiplied by `read_pref_io`. Generally, any multiple or factor of `read_nstream` multiplied by `read_pref_io` should be a good size for performance. For writing, the same rule of thumb applies to the `write_pref_io` and `write_nstream` parameters. When tuning a file system, the best thing to do is try out the tuning parameters under a real life workload.

If an application is doing sequential I/O to large files, it should try to issue requests larger than the `discovered_direct_iosz`. This causes the I/O requests to be performed as discovered direct I/O requests, which are unbuffered like direct I/O but do not require synchronous inode updates when extending the file. If the file is larger than can fit in the cache, using unbuffered I/O avoids removing useful data out of the cache and lessens CPU overhead.

# Extent Attributes 3

The VERITAS File System (VxFS) allocates disk space to files in groups of one or more adjacent blocks called *extents*. VxFS defines an application interface that allows programs to control various aspects of the extent allocation for a given file (see "Extent Information" on page 57). The extent allocation policies associated with a file are referred to as *extent attributes*.

The VxFS `getext` and `setext` commands let you view or manipulate file extent attributes. In addition, the `vxdump`, `vxrestore`, `mv`, `cp`, and `cpio` commands preserve extent attributes when a file is backed up, moved, copied, or archived.

The following topics are covered in this chapter:





# Attribute Specifics

The two basic extent attributes associated with a file are its *reservation* and its *fixed extent size*. You can preallocate space to the file by manipulating a file's reservation, or override the default allocation policy of the file system by setting a fixed extent size.

Other policies determine the way these attributes are expressed during the allocation process. You can specify that:

- The space reserved for a file must be contiguous
- No allocations are made for a file beyond the current reservation
- An unused reservation is released when the file is closed
- Space is allocated, but no reservation is assigned
- The file size is changed to immediately incorporate the allocated space

Some of the extent attributes are persistent and become part of the on-disk information about the file, while other attributes are temporary and are lost after the file is closed or the system is rebooted. The persistent attributes are similar to the file's permissions and are written in the inode for the file. When a file is copied, moved, or archived, only the persistent attributes of the source file are preserved in the new file (see "Other Controls" on page 50 for more information).

In general, the user will only set extent attributes for reservation. Many of the attributes are designed for applications that are tuned to a particular pattern of I/O or disk alignment (see the `mkfs_vxfs`(1M) manual page and "Application Interface" on page 53 for more information).

## Reservation: Preallocating Space to a File

VxFS makes it possible to preallocate space to a file at the time of the request rather than when data is written into the file. This space cannot be allocated to other files in the file system. VxFS prevents any unexpected out-of-space condition on the file system by ensuring that a file's required space will be associated with the file before it is required.

A persistent reservation is not released when a file is truncated. The reservation must be cleared or the file must be removed to free the reserved space.

## Fixed Extent Size

The VxFS default allocation policy uses a variety of methods to determine how to make an allocation to a file when a write requires additional space. The policy attempts to balance the two goals of optimum I/O performance through large allocations and minimal file system fragmentation through allocation from space available in the file system that best fits the data.

Setting a fixed extent size overrides the default allocation policies for a file and always serves as a persistent attribute. Be careful to choose an extent size appropriate to the application when using fixed extents. An advantage of VxFS's extent-based allocation policies is that they rarely use indirect blocks compared to block based file systems; VxFS eliminates many instances of disk access that stem from indirect references. However, a small extent size can eliminate this advantage.

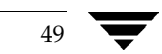
Files with aggressive allocation sizes tend to be more contiguous and have better I/O characteristics. However, the overall performance of the file system degrades because the unused space fragments free space by breaking large extents into smaller pieces. By erring on the side of minimizing fragmentation for the file system, files may become so non-contiguous that their I/O characteristics would degrade.

Fixed extent sizes are particularly appropriate in the following situations:

- If a file is large and sparse and its write size is fixed, a fixed extent size that is a multiple of the write size can minimize space wasted by blocks that do not contain user data as a result of misalignment of write and extent sizes.
- If a file is large and contiguous, a large fixed extent size can minimize the number of extents in the file.

Custom applications may also use fixed extent sizes for specific reasons, such as the need to align extents to cylinder or striping boundaries on disk.





## Other Controls

The auxiliary controls on extent attributes determine:

- Whether allocations are aligned
- Whether allocations are contiguous
- Whether the file can be written beyond its reservation
- Whether an unused reservation is released when the file is closed
- Whether the reservation is a persistent attribute of the file
- When the space reserved for a file will actually become part of the file

### Alignment

Specific alignment restrictions coordinate a file's allocations with a particular I/O pattern or disk alignment (see the `mkfs_vxfs`(1M) manual page and "Application Interface" on page 53 for details). Alignment can only be specified if a fixed extent size has also been set. Setting alignment restrictions on allocations is best left to well designed applications.

### Contiguity

A reservation request can specify that its allocation remain contiguous (all one extent). Maximum contiguity of a file optimizes its I/O characteristics.

**Note** Fixed extent sizes or alignment cause a file system to return an error message reporting insufficient space if no suitably sized (or aligned) extent is available. This can happen even if the file system has sufficient free space and the fixed extent size is large.

### Write Operations Beyond Reservation

A reservation request can specify that no allocations can take place after a write operation fills up the last available block in the reservation. This specification can be used in a similar way to `ulimit` to prevent a file's uncontrolled growth.

### Reservation Trimming

A reservation request can specify that any unused reservation be released when the file is closed. The file is not completely closed until all processes open against the file have closed it.

### Reservation Persistence

A reservation request can ensure that the reservation does not become a persistent attribute of the file. The unused reservation is discarded when the file is closed.

### Including Reservation in the File

A reservation request can make sure the size of the file is adjusted to include the reservation. Normally, the space of the reservation is not included in the file until an extending write operation requires it. A reservation that immediately changes the file size can generate large temporary files. Unlike a `ftruncate` operation that increases the size of a file, this type of reservation does not perform zeroing of the blocks included in the file and limits this facility to users with appropriate privileges. The data that appears in the file may have been previously contained in another file.

## Commands Related to Extent Attributes

The VxFS commands for manipulating extent attributes are `setext` and `getext`; they allow the user to set up files with a given set of extent attributes or view any attributes that are already associated with a file. See the `setext`(1M) and `getext`(1M) manual pages for details on using these commands.

The VxFS-specific commands `vxdump` and `vxrestore`, and the `mv`, `cp`, and `cpio` commands, preserve extent attributes when backing up, restoring, moving, or copying files.

Most of these commands include a command line option (`-e`) for maintaining extent attributes on files. This option specifies dealing with a VxFS file that has extent attribute information including reserved space, a fixed extent size, and extent alignment. The extent attribute information may be lost if the destination file system does not support extent attributes, has a different block size than the source file system, or lacks free extents appropriate to satisfy the extent attribute requirements.

The `-e` option takes any of the following keywords as an argument:

| | |
|---|---|
| `warn` | Issues a warning message if extent attribute information cannot be maintained (the default) |
| `force` | Fails the copy if extent attribute information cannot be maintained |
| `ignore` | Ignores extent attribute information entirely |



## Failure to Preserve Extent Attributes

Whenever a file is copied, moved, or archived using commands that preserve extent attributes, there is nevertheless the possibility of losing the attributes. Such a failure might occur for three reasons:

- The file system receiving a copied, moved, or restored file from an archive is not a VxFS type. Since other file system types do not support the extent attributes of the VxFS file system, the attributes of the source file are lost during the migration.
- The file system receiving a copied, moved, or restored file is a VxFS type but does not have enough free space to satisfy the extent attributes. For example, consider a 50K file and a reservation of 1 MB. If the target file system has 500K free, it could easily hold the file but fail to satisfy the reservation.
- The file system receiving a copied, moved, or restored file from an archive is a VxFS type but the different block sizes of the source and target file system make extent attributes impossible to maintain. For example, consider a source file system of block size 1024, a target file system of block size 4096, and a file that has a fixed extent size of 3 blocks (3072 bytes). This fixed extent size adapts to the source file system but cannot translate onto the target file system.

  The same source and target file systems in the preceding example with a file carrying a fixed extent size of 4 could preserve the attribute; a 4 block (4096 byte) extent on the source file system would translate into a 1 block extent on the target.

  On a system with mixed block sizes, a copy, move, or restoration operation may or may not succeed in preserving attributes. It is recommended that the same block size be used for all file systems on a given system.

# Application Interface 4

The VERITAS File System (VxFS) provides enhancements that can be used by applications that require certain performance features. This chapter describes cache advisories and provides information about fixed extent sizes and reservation of space for a file.

If you are writing applications, you can optimize them for use with the VxFS. To optimize VxFS for use with applications, see "VxFS Performance: Creating, Mounting, and Tuning File Systems" on page 21.

The following topics are covered in this chapter:





# Cache Advisories

VxFS allows an application to set cache advisories for use when accessing files. These advisories are in memory only and they do not persist across reboots. Some advisories are currently maintained on a per-file, not a per-file-descriptor, basis. This means that only one set of advisories can be in effect for all accesses to the file. If two conflicting applications set different advisories, both use the last advisories that were set.

All advisories are set using the `VX_SETCACHE` ioctl command. The current set of advisories can be obtained with the `VX_GETCACHE` ioctl command. For details on the use of these ioctl commands, see the `vxfsio`(7) manual page.

## Direct I/O

Direct I/O is an unbuffered form of I/O. If the `VX_DIRECT` advisory is set, the user is requesting direct data transfer between the disk and the user-supplied buffer for reads and writes. This bypasses the kernel buffering of data, and reduces the CPU overhead associated with I/O by eliminating the data copy between the kernel buffer and the user's buffer. This also avoids taking up space in the buffer cache that might be better used for something else. The direct I/O feature can provide significant performance gains for some applications.

For an I/O operation to be performed as direct I/O, it must meet certain alignment criteria. The alignment constraints are usually determined by the disk driver, the disk controller, and the system memory management hardware and software. The requirements for direct I/O are as follows:

- The starting file offset must be aligned to a 512-byte boundary.
- The ending file offset must be aligned to a 512-byte boundary, or the length must be a multiple of 512 bytes.
- The memory buffer must start on an 8-byte boundary.

If the I/O is performed using the `readv`(2) and `writev`(2) system calls, these restrictions apply to each element of the array of `struct iovec`.

The requirements to perform direct I/O on a given platform and operating system release may be less restrictive than above, but these requirements are met, then direct I/O will work on any platform. In particular, Solaris and HP-UX do not require any alignment of the memory buffer.

Also note that on HP-UX, direct I/O will be the most efficient if the starting and ending file offsets are aligned on file system block boundaries, as reported in the field `f_frsize` of `statvfs`(2).

If a request fails to meet the alignment constraints for direct I/O, the request is performed as data synchronous I/O. If the file is currently being accessed by using memory mapped I/O, any direct I/O accesses are done as data synchronous I/O.

Because direct I/O maintains the same data integrity as synchronous I/O, it can be used in many applications that currently use synchronous I/O. If a direct I/O request does not allocate storage or extend the file, the inode is not immediately written.

The CPU cost of direct I/O is about the same as a raw disk transfer. For sequential I/O to very large files, using direct I/O with large transfer sizes can provide the same speed as buffered I/O with much less CPU overhead.

If the file is being extended or storage is being allocated, direct I/O must write the inode change before returning to the application. This eliminates some of the performance advantages of direct I/O.

The direct I/O and `VX_DIRECT` advisories are maintained on a per-file-descriptor basis.

## Unbuffered I/O

If the `VX_UNBUFFERED` advisory is set, I/O behavior is the same as direct I/O with the `VX_DIRECT` advisory set, so the alignment constraints that apply to direct I/O also apply to unbuffered I/O. For unbuffered I/O, however, if the file is being extended, or storage is being allocated to the file, inode changes are not updated synchronously before the write returns to the user. The `VX_UNBUFFERED` advisory is maintained on a per-file-descriptor basis.

## Discovered Direct I/O

Discovered Direct I/O is a file system tunable you can set using the `vxtunefs` command. When the file system gets an I/O request larger than the `discovered_direct_iosz`, it tries to use direct I/O on the request. For large I/O sizes, Discovered Direct I/O can perform much better than buffered I/O.

Discovered Direct I/O behavior is similar to direct I/O and has the same alignment constraints, except writes that allocate storage or extend the file size do not require writing the inode changes before returning to the application.

For information on how to set the `discovered_direct_iosz`, see "I/O Tuning" on page 37.



## Data Synchronous I/O

If the VX_DSYNC advisory is set, the user is requesting data synchronous I/O. In synchronous I/O, the data is written, and the inode is written with updated times and (if necessary) an increased file size. In data synchronous I/O, the data is transferred to disk synchronously before the write returns to the user. If the file is not extended by the write, the times are updated in memory, and the call returns to the user. If the file is extended by the operation, the inode is written before the write returns.

Like direct I/O, the data synchronous I/O feature can provide significant application performance gains. Because data synchronous I/O maintains the same data integrity as synchronous I/O, it can be used in many applications that currently use synchronous I/O. If the data synchronous I/O does not allocate storage or extend the file, the inode is not immediately written. The data synchronous I/O does not have any alignment constraints, so applications that find it difficult to meet the alignment constraints of direct I/O should use data synchronous I/O.

If the file is being extended or storage is allocated, data synchronous I/O must write the inode change before returning to the application. This case eliminates the performance advantage of data synchronous I/O.

The direct I/O and VX_DSYNC advisories are maintained on a per-file-descriptor basis.

## Other Advisories

The VX_SEQ advisory indicates that the file is being accessed sequentially. When the file is being read, the maximum read-ahead is always performed. When the file is written, instead of trying to determine whether the I/O is sequential or random by examining the write offset, sequential I/O is assumed. The buffers for the write are not immediately flushed. Instead, buffers are flushed some distance behind the current write point.

The VX_RANDOM advisory indicates that the file is being accessed randomly. For reads, this disables read-ahead. For writes, this disables the flush-behind. The data is flushed at a rate based on memory contention.

The VX_NOREUSE advisory is used as a modifier. If both VX_RANDOM and VX_NOREUSE are set, VxFS notifies the operating system that the buffers are free and may be reclaimed. If VX_NOREUSE is set when doing sequential I/O, buffers are also freed when they are flushed to disk. The VX_NOREUSE advisory may slow down access to the file, but it can reduce the cached data held by the system. This can allow more data to be cached for other files and may speed up those accesses.

# Extent Information

The VX_SETEXT ioctl command allows an application to reserve space for a file, and set fixed extent sizes and file allocation flags. Applications can obtain status information on VxFS files by using the VX_GETEXT ioctl. The `getext` command also provides access to this information. See the `setext`(1M), `getext`(1M), and `vxfsio`(7) manual pages for more information.

Each invocation of the VX_SETEXT ioctl affects all the elements in the `vx_ext` structure. When using VX_SETEXT, always use the following procedure:

### ▼ To use VX_SETEXT

1. Use VX_GETEXT to read the current settings.
2. Modify the values to be changed.
3. Call VX_SETEXT to set the values.

**Caution** Follow this procedure carefully. A fixed extent size may be inadvertently cleared when the reservation is changed.



## Space Reservation

Storage can be reserved for a file at any time. When a VX_SETEXT ioctl is issued, the reservation value is set in the inode on disk. If the file size is less than the reservation amount, the kernel allocates space to the file from the current file size up to the reservation amount. When the file is truncated, space below the reserved amount is not freed. The VX_TRIM, VX_NOEXTEND, VX_CHGSIZE, VX_NORESERVE and VX_CONTIGUOUS flags can be used to modify reservation requests.

**Note** VX_NOEXTEND is the only one of these flags that is persistent; the other flags may have persistent effects, but they are not returned by the VX_GETEXT ioctl.

If the VX_TRIM flag is set, when the last close occurs on the inode, the reservation is trimmed to match the file size and the VX_TRIM flag is cleared. Any unused space is freed. This can be useful if an application needs enough space for a file, but it is not known how large the file will become. Enough space can be reserved to hold the largest expected file, and when the file has been written and closed, any extra space will be released.

If the VX_NOEXTEND flag is set, an attempt to write beyond the current reservation, which requires the allocation of new space for the file, fails instead. To allocate new space to the file, the space reservation must be increased. This can be used like ulimit to prevent a file from using too much space.

If the VX_CONTIGUOUS flag is set, any space allocated to satisfy the current reservation request is allocated in one extent. If there is not one extent large enough to satisfy the request, the request fails. For example, if a file is created and a 1 MB contiguous reservation is requested, the file size is set to zero and the reservation to 1 MB. The file will have one extent that is 1 MB long. If another reservation request is made for a 3 MB contiguous reservation, the new request will find that the first 1 MB is already allocated and allocate a 2 MB extent to satisfy the request. If there are no 2 MB extents available, the request fails. Extents are, by definition, contiguous.

**Note** Because VX_CONTIGUOUS is not a persistent flag, space will not be allocated contiguously after doing a file system restore.

If the VX_NORESERVE flag is set, the reservation value in the inode is not changed. This flag is used by applications to do temporary reservation. Any space past the end of the file is given up when the file is closed. For example, if the cp command is copying a file that is 1 MB long, it can request a 1 MB reservation with the VX_NORESERVE flag set. The space is allocated, but the reservation in the file is left at 0. If the program aborts for any reason or the system crashes, the unused space past the end of the file is released. When the program finishes, there is no cleanup because the reservation was never recorded on disk.

If the VX_CHGSIZE flag is set, the file size is increased to match the reservation amount. This flag can be used to create files with uninitialized data. Because this allows uninitialized data in files, it is restricted to users with appropriate privileges.

It is possible to use these flags in combination. For example, using VX_CHGSIZE and VX_NORESERVE changes the file size but does not set any reservation. When the file is truncated, the space is freed. If the VX_NORESERVE flag had not been used, the reservation would have been set on disk along with the file size.

Space reservation is used to make sure applications do not fail because the file system is out of space. An application can preallocate space for all the files it needs before starting to do any work. By allocating space in advance, the file is optimally allocated for performance, and file accesses are not slowed down by the need to allocate storage. This allocation of resources can be important in applications that require a guaranteed response time.

With very large files, use of space reservation can avoid the need to use indirect extents. It can also improve performance and reduce fragmentation by guaranteeing that the file consists of large contiguous extents. Sometimes when critical file systems run out of space, cron jobs, mail, or printer requests fail. These failures are harder to track if the logs kept by the application cannot be written due to a lack of space on the file system.

By reserving space for key log files, the logs will not fail when the system runs out of space. Process accounting files can also have space reserved so accounting records will not be lost if the file system runs out of space. In addition, by using the VX_NOEXTEND flag for log files, the maximum size of these files can be limited. This can prevent a runaway failure in one component of the system from filling the file system with error messages and causing other failures. If the VX_NOEXTEND flag is used for log files, the logs should be cleaned up before they reach the size limit in order to avoid losing information.



## Fixed Extent Sizes

VxFS uses the I/O size of write requests, and a default policy, when allocating space to a file. For some applications, this may not work out well. These applications can set a fixed extent size, so that all new extents allocated to the file are of the fixed extent size.

By using a fixed extent size, an application can reduce allocations and guarantee good extent sizes for a file. An application can reserve most of the space a file needs, and then set a relatively large fixed extent size. If the file grows beyond the reservation, any new extents are allocated in the fixed extent size.

Another use of a fixed extent size occurs with sparse files. The file system usually does I/O in page size multiples. When allocating to a sparse file, the file system allocates pages as the smallest default unit. If the application always does sub-page I/O, it can request a fixed extent size to match its I/O size and avoid wasting extra space.

When setting a fixed extent size, an application should not select too large a size. When all extents of the required size have been used, attempts to allocate new extents fail: this failure can happen even though there are blocks free in smaller extents.

Fixed extent sizes can be modified by the `VX_ALIGN` flag. If the `VX_ALIGN` flag is set, then any future extents allocated to the file are aligned on a fixed extent size boundary relative to the start of the allocation unit. This can be used to align extents to disk striping boundaries or physical disk boundaries.

The `VX_ALIGN` flag is persistent and is returned by the `VX_GETEXT` ioctl.

# Freeze and Thaw

The `VX_FREEZE` ioctl command is used to freeze a file system. Freezing a file system temporarily blocks all I/O operations to a file system and then performs a `sync` on the file system. When the `VX_FREEZE` ioctl is issued, all access to the file system is blocked at the system call level. Current operations are completed and the file system is synchronized to disk. Freezing provides a stable, consistent file system.

When the file system is frozen, any attempt to use the frozen file system, except for a `VX_THAW` ioctl command, is blocked until a process executes the `VX_THAW` ioctl command or the time-out on the freeze expires.

# Get I/O Parameters ioctl

VxFS provides the `VX_GET_IOPARAMETERS` ioctl to get the recommended I/O sizes to use on a file system. This ioctl can be used by the application to make decisions about the I/O sizes issued to VxFS for a file or file device. For more details on this ioctl, refer to the `vxfsio`(7) manual page. For a discussion on various I/O parameters, refer to "VxFS Performance: Creating, Mounting, and Tuning File Systems" on page 21 and the `vxtunefs`(1M) manual page.

# Named Data Streams

Named data streams associate multiple data streams with a file. Access to the named data stream can be done through a file descriptor using the named data stream library functions. Applications can open the named data stream to obtain a file descriptor and perform `read`(), `write`(), and `mmap`() operations using the file descriptor. These system calls would work as though they are operating on a regular file. The named data streams of a file are stored in a hidden named data stream directory inode associated with the file. The hidden directory inode for the file can be accessed only through the named data stream application programming interface.

**Note** Named data streams are also known as named attributes.



# Named Data Streams Programmatic Interface

VxFS named data stream functionality is available only through the following application programming interface (API) functions:

| | |
|---|---|
| vxfs_nattr_link | Links to a named data stream. |
| vxfs_nattr_open | Open a named data stream. |
| vxfs_nattr_rename | Renames a named data stream. |
| vxfs_nattr_unlink | Removes a named data stream. |

The `vxfs_nattr_open`() function works similarly to the `open`() system call, except that the path is interpreted as a named data stream to a file descriptor. If the `vxfs_nattr_open`() operation completes successfully, the return value is the file descriptor associated with the named data stream. The file descriptor can be used by other input/output functions to refer to that named data stream. If the path of the named data stream is set to "`.`" the file descriptor returned points to the named data stream directory vnode.

The `vxfs_nattr_link`() function creates a new directory entry for the existing named data stream and increments its link count by one. There is a pointer to an existing named data stream in the named data stream namespace and a pointer to the new directory entry created in the named data stream namespace.

The `vxfs_nattr_unlink`() function removes the named data stream at a specified path. The calling function must have write permission to remove the directory entry for the named data stream.

The `vxfs_nattr_rename`() function changes a specified namespace entry at `path1` to a second specified namespace at `path2`. The specified paths are resolved relative to a pointer to the named data stream directory vnodes.

See the `vxfs_nattr_open`(3), `vxfs_nattr_link`(3), `vxfs_nattr_unlink`(3), and `vxfs_nattr_rename`(3) manual pages for more information.

## Listing Named Data Streams

The named data streams for a file can be listed by calling `getdents`() on the named data stream directory inode. For example:

```
fd = open("foo", O_RDWR);           /* open file foo */
afd = vxfs_nattr_open(fd, "attribute1",
             O_RDWR|O_CREAT, 0777);  /* create attribute
                                        attribute1 for file foo */
write(afd, buf, 1024);              /* writes to attribute file */
read(afd, buf, 1024);               /* reads from attribute file */
dfd = vxfs_nattr_open(fd, ".", O_RDONLY);/* opens attribute
                                            directory for file foo */
getdents(dfd, buf, 1024);           /* reads directory entries for
                                        attribute directory */
```

## Namespace for Named Data Streams

Names starting with "`$vxfs:`" are reserved for future use. Creating a data stream where the name starts with "`$vxfs:`" fail with an EINVAL error.

## Behavior Changes in Other System Calls

Though the named data stream directory is hidden from the namespace, it is possible to open the name data stream directory inode with a `fchdir`() or `fchroot`() call. Some of the attributes (such as "`..`") are not defined for a named data streams directory. Any operation that accesses these fields can fail. Attempts to create directories, symbolic links, or device files on a named data stream directory will fail. VOP_SETATTR() done on a named data stream directory or named data stream inode will also fail.



# Storage Checkpoints 5

Storage Checkpoints are a feature of the VERITAS File System (VxFS) that provide *point-in-time* images of file system contents. These frozen images of VxFS file systems can be used in a variety of applications such as full and incremental online backups, fast error recovery, and product development testing. Storage Checkpoint replicas of real time databases can also be used for decision support and an assortment of database analyses.

The following topics are covered in this chapter:





# What is a Storage Checkpoint?

The VERITAS File System provides a unique Storage Checkpoint facility that quickly creates a persistent image of a file system at an exact point in time. Storage Checkpoints significantly reduce I/O overhead by identifying and maintaining only the file system blocks that have changed since the last Storage Checkpoint or backup via a *copy-on-write* technique (see "How a Storage Checkpoint Works" on page 67). Unlike a disk-based mirroring technology that requires a separate storage space, this VERITAS technology minimizes the use of disk space by creating a Storage Checkpoint within the same free space available to the file system.

Storage Checkpoints are data objects that are managed and controlled by the file system; as a result, Storage Checkpoints are persistent across system reboots and crashes. You can create, remove, and rename Storage Checkpoints because they are data objects with associated names (see "Storage Checkpoint Administration" on page 71). After you create a Storage Checkpoint of a mounted file system, you can also continue to create, remove, and update files on the file system without affecting the logical image of the Storage Checkpoint. This technology preserves not only the name space (directory hierarchy) of the file system, but also the user data as it existed at the moment the Storage Checkpoint was taken.

Storage Checkpoints differ from VERITAS File System snapshots in the following ways because they:

- Allow write operations to the Storage Checkpoint itself.
- Persist after a system reboot or failure.
- Share the same pool of free space as the file system.
- Maintain a relationship with other Storage Checkpoints by identifying changed file blocks since the last Storage Checkpoint.
- Have multiple, read-only Storage Checkpoints that reduce I/O operations and required storage space because the most recent Storage Checkpoint is the only one that accumulates updates from the primary file system.

Various backup and replication solutions can take advantage of Storage Checkpoints. The ability of Storage Checkpoints to track the file system blocks that have changed since the last Storage Checkpoint facilitates backup and replication applications that only need to retrieve the changed data. Storage Checkpoints significantly minimize data movement and may promote higher availability and data integrity by increasing the frequency of backup and replication solutions.

Storage Checkpoints can be taken in environments with a large number of files (for example, file servers with millions of files) with little adverse impact on performance. Because the file system does not remain frozen during Storage Checkpoint creation, applications can access the file system even while the Storage Checkpoint is taken. However, Storage Checkpoint creation may take several minutes to complete depending on the number of files in the file system.

## How a Storage Checkpoint Works

The Storage Checkpoint facility freezes the mounted file system (known as the *primary fileset*), initializes the Storage Checkpoint, and thaws the file system. Specifically, the file system is first brought to a stable state where all of its data is written to disk, and the freezing process momentarily blocks all I/O operations to the file system. A Storage Checkpoint is then created without any actual data; the Storage Checkpoint instead points to the *block map* (described below) of the primary fileset. The *thawing* process that follows restarts I/O operations to the file system.

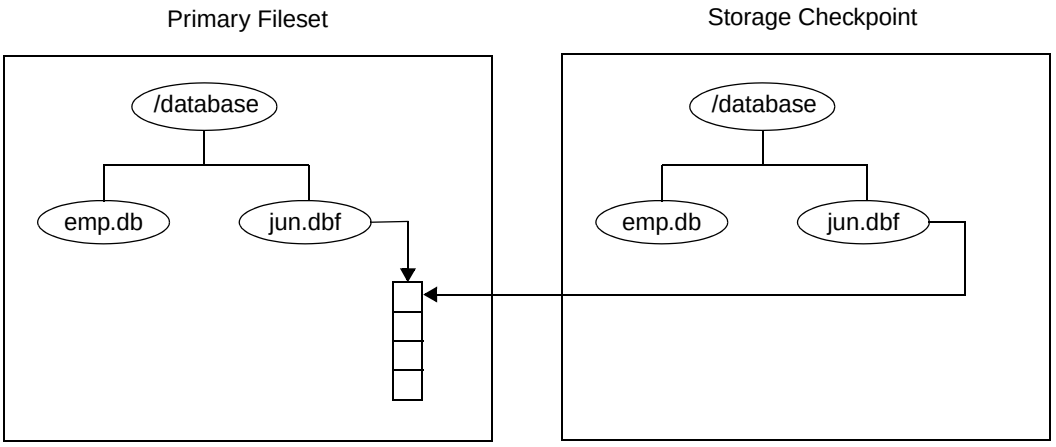
You can create a Storage Checkpoint on a single file system or a list of file systems. A Storage Checkpoint of multiple file systems simultaneously freezes the file systems, creates a Storage Checkpoint on all of the file systems, and thaws the file systems. As a result, the Storage Checkpoints for multiple file systems have the same creation timestamp. The Storage Checkpoint facility guarantees that multiple file system Storage Checkpoints are created on all or none of the specified file systems (unless there is a system crash while the operation is in progress).

**Note** The calling application is responsible for cleaning up Storage Checkpoints after a system crash.

As mentioned above, a Storage Checkpoint of the primary fileset initially contains a pointer to the file system block map rather than to any actual data. The block map points to the data on the primary fileset. The figure below shows the file system `/database` and its Storage Checkpoint. The Storage Checkpoint is logically identical to the primary fileset when the Storage Checkpoint is created, but it does not contain any actual data blocks.

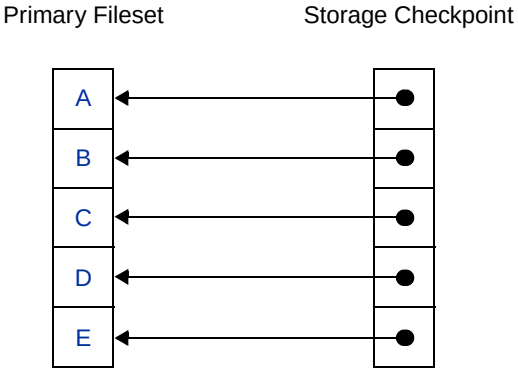


Primary Fileset and Its Storage Checkpoint




In the figure below, each block of the file system is represented by a square. Similar to the previous figure, this figure shows a Storage Checkpoint containing pointers to the primary fileset at the time the Storage Checkpoint is taken.

Initializing a Storage Checkpoint



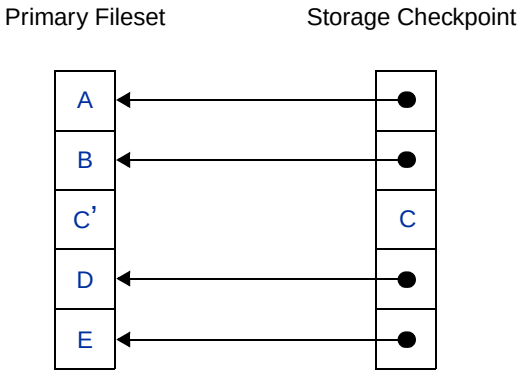

The Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. As the primary fileset is updated, the original data is copied to the Storage Checkpoint before the new data is written. When a write operation changes a specific data block in the primary fileset, the old data is first read and copied to the Storage Checkpoint before the primary fileset is updated. Subsequent writes to the

specified data block on the primary fileset do not result in additional updates to the Storage Checkpoint because the old data needs to be saved only once. As blocks in the primary fileset continue to change, the Storage Checkpoint accumulates the original data blocks.

In the following figure, the third block originally containing C is updated. Before the block is updated with new data, the original data is copied to the Storage Checkpoint. This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken.

Every update or write operation does not necessarily result in the process of copying data to the Storage Checkpoint. In this example, subsequent updates to this block, now containing C', are not copied to the Storage Checkpoint because the original image of the block containing C is already saved.

Updates to the Primary Fileset



# Types of Storage Checkpoints

You can create the following types of Storage Checkpoints:

- Data Storage Checkpoints
- Nodata Storage Checkpoints
- Removable Storage Checkpoints
- Non-Mountable Storage Checkpoints

## Data Storage Checkpoints

A data Storage Checkpoint is a complete image of the file system at the time the Storage Checkpoint is created. This type of Storage Checkpoint contains the file system metadata and file data blocks. You can mount, access, and write to a data Storage Checkpoint just as you would to a file system. Data Storage Checkpoints are useful for backup applications that require a consistent and stable image of an active file system. Data Storage Checkpoints introduce some overhead to the system and to the application performing the write operation. For best results, limit the life of data Storage Checkpoints to minimize the impact on system resources. See "Difference Between a Data and a Nodata Storage Checkpoint" on page 76.

## Nodata Storage Checkpoints

A nodata Storage Checkpoint only contains file system metadata—no file data blocks. As the original file system changes, the nodata Storage Checkpoint records the location of every changed block. Nodata Storage Checkpoints use minimal system resources and have little impact on the performance of the file system because the data itself does not have to be copied. See "Difference Between a Data and a Nodata Storage Checkpoint" on page 76.

## Removable Storage Checkpoints

A removable Storage Checkpoint can "self-destruct" under certain conditions when the file system runs out of space (see "Space Management Considerations" on page 82 for more information). After encountering certain out-of-space (ENOSPC) conditions, the kernel removes Storage Checkpoints to free up space for the application to continue running on the file system. In almost all situations, you should create Storage Checkpoints with the removable attribute.

### Non-Mountable Storage Checkpoints

A non-mountable Storage Checkpoint cannot be mounted. You can use this type of Storage Checkpoint as a security feature which prevents other applications from accessing the Storage Checkpoint and modifying it.

## Storage Checkpoint Administration

Storage Checkpoint administrative operations require the `fsckptadm` utility (see the `fsckptadm`(1M) manual page). You can use the `fsckptadm` utility to create and remove Storage Checkpoints, change attributes, and ascertain statistical data. Every Storage Checkpoint has an associated name, which allows you to manage Storage Checkpoints; this name is limited to 127 characters and cannot contain a colon (`:`).

Storage Checkpoints require some space for metadata on the volume or set of volumes specified by the file system allocation policy or Storage Checkpoint allocation policy. The `fsckptadm` utility displays an error if the volume or set of volumes does not have enough free space to contain the metadata. You can roughly approximate the amount of space required by the metadata using a method that depends on the disk layout version of the file system.

For disk layout Version 5 or prior, multiply the number of inodes (# of inodes) by the inode size (inosize) in bytes, and add 1 or 2 megabytes to get the approximate amount of space required. You can determine the number of inodes with the `fsckptadm` utility, and the inode size with the `mkfs` command:

```
# fsckptadm -v info '' /mnt0
UNNAMED:
        ctime                =  Thu 3 Mar 2005 7:00:17 PM PST
        mtime                =  Thu 3 Mar 2005 7:00:17 PM PST
        flags                =  largefiles, mounted
        # of inodes          =  23872
        # of blocks          =  27867
        .
        .
        .
        # of overlay bmaps   =  0
# mkfs -m /mnt0
mkfs -F vxfs -o bsize=1024,version=6,inosize=256,logsize=16384,
  largefiles /mnt0
```

In this example, the approximate amount of space required by the metadata is 7 or 8 megabytes (23,872 x 256 bytes, plus 1 or 2 megabytes).



For disk layout Version 6, multiply the number of inodes by 1 byte, and add 1 or 2 megabytes to get the approximate amount of space required. You can determine the number of inodes with the `fsckptadm` utility as above. Using the output from the example for disk layout Version 5 or prior, the approximate amount of space required by the metadata is just over one or two megabytes (23,872 x 1 byte, plus 1 or 2 megabytes).

Use the `fsvoladm` command to determine if the volume set has enough free space (see the `fsvoladm`(1M) manual page):

```
# fsvoladm list /mnt0
devid   size       used     avail      name
0       20971520   8497658  12473862   mnt1
1       20971520   6328993  14642527   mnt2
2       20971520   4458462  16513058   mnt3
```

## Creating a Storage Checkpoint

You can create a Storage Checkpoint using the `fsckptadm` utility. In these examples, `/mnt0` is a mounted VxFS file system with a Version 4 or Version 5 disk layout.

This example shows the creation of a nodata Storage Checkpoint (see "Space Management Considerations" on page 82) named `thu_7pm` on `/mnt0` and lists all Storage Checkpoints of the `/mnt0` file system:

```
# fsckptadm -n create thu_7pm /mnt0
# fsckptadm list /mnt0
/mnt0
thu_7pm:
  ctime= Thu 3 Mar 2005 7:00:17 PM PST
  mtime= Thu 3 Mar 2005 7:00:17 PM PST
  flags= nodata, largefiles
```

This example shows the creation of a removable Storage Checkpoint named `thu_8pm` on `/mnt0` and lists all Storage Checkpoints of the `/mnt0` file system:

```
# fsckptadm -r create thu_8pm /mnt0
# fsckptadm list /mnt0
/mnt0
thu_8pm:
  ctime= Thu 3 Mar 2005 8:00:19 PM PST
  mtime= Thu 3 Mar 2005 8:00:19 PM PST
  flags= largefiles, removable
thu_7pm:
  ctime= Thu 3 Mar 2005 7:00:17 PM PST
  mtime= Thu 3 Mar 2005 7:00:17 PM PST
  flags= nodata, largefiles
```

## Removing a Storage Checkpoint

You can delete a Storage Checkpoint by specifying the `remove` keyword of the `fsckptadm` command. Specifically, you can use either the *synchronous* or *asynchronous* method of removing a Storage Checkpoint; the asynchronous method is the default method. The synchronous method entirely removes the Storage Checkpoint and returns all of the blocks to the file system before completing the `fsckptadm` operation. The asynchronous method simply marks the Storage Checkpoint for removal and causes `fsckptadm` to return immediately. At a later time, an independent kernel thread completes the removal operation and releases the space used by the Storage Checkpoint.

In this example, `/mnt0` is a mounted VxFS file system with a Version 4 disk layout. This example shows the asynchronous removal of the Storage Checkpoint named `thu_8pm` and synchronous removal of the Storage Checkpoint named `thu_7pm`. This example also lists all the Storage Checkpoints remaining on the `/mnt0` file system after the specified Storage Checkpoint is removed:

```
# fsckptadm remove thu_8pm /mnt0
# fsckptadm list /mnt0
/mnt0
thu_7pm:
  ctime= Thu 3 Mar 2005 7:00:17 PM PST
  mtime= Thu 3 Mar 2005 7:00:17 PM PST
  flags= nodata, largefiles
# fsckptadm -s remove thu_7pm /mnt0
# fsckptadm list /mnt0
/mnt0
```

## Accessing a Storage Checkpoint

You can mount Storage Checkpoints using the `mount` command (see the `mount_vxfs`(1M) manual page) with the mount option `–o ckpt=ckpt_name`. Observe the following rules when mounting Storage Checkpoints:

- Storage Checkpoints are mounted as read-only Storage Checkpoints by default. If you need to write to a Storage Checkpoint, mount it using the `-o rw` option.
- If a Storage Checkpoint is originally mounted as a read-only Storage Checkpoint, you can remount it as a writable Storage Checkpoint using the `-o remount` option.
- To mount a Storage Checkpoint of a file system, first mount the file system itself.
- To unmount a file system, first unmount all of its Storage Checkpoints.

**Caution** If you create a Storage Checkpoint for backup purposes, do not mount it as a writable Storage Checkpoint. You will lose the point-in-time image if you accidently write to the Storage Checkpoint.



A Storage Checkpoint is mounted on a special *pseudo device*. This pseudo device does not exist in the system name space; the device is internally created by the system and used while the Storage Checkpoint is mounted. The pseudo device is removed after you unmount the Storage Checkpoint. A pseudo device name is formed by appending the Storage Checkpoint name to the file system device name using the colon character (`:`) as the separator.

For example, if a Storage Checkpoint named `may_23` belongs to the file system residing on the special device `/dev/vx/dsk/fsvol/vol1`, the Storage Checkpoint pseudo device name is:

```
/dev/vx/dsk/fsvol/vol1:may_23
```

To mount the Storage Checkpoint named `may_23` as a read-only (default) Storage Checkpoint on directory `/fsvol_may_23`, type:

```
# mount -F vxfs -o ckpt=may_23 /dev/vx/dsk/fsvol/vol1:may_23 \
 /fsvol_may_23
```

The `/fsvol` file system must already be mounted before the Storage Checkpoint can be mounted. To remount the Storage Checkpoint named `may_23` as a writable Storage Checkpoint, type:

```
# mount -F vxfs -o ckpt=may_23,remount,rw \
  /dev/vx/dsk/fsvol/vol1:may_23 /fsvol_may_23
```

To mount this Storage Checkpoint automatically when the system starts up, put the following entries in the `/etc/fstab` file:

| Device Special File | device to fsck | mount point | FS type | fsck pass | mount at boot | mount options |
|---|---|---|---|---|---|---|
| /dev/vx/dsk/ fsvol/vol1 | /dev/vx/rdsk/ fsvol/vol1 | /fsvol | vxfs | 1 | yes | — |
| /dev/vx/dsk/fsvol/ vol1:may_23 | — | /fsvol_may_23 | vxfs | 0 | yes | ckpt= may_23 |

To mount a Storage Checkpoint of a cluster file system, you must also use the `-o cluster` option:

```
# mount -F vxfs -o cluster,ckpt=may_23 \
  /dev/vx/dsk/fsvol/vol1:may_23 /fsvol_may_23
```

You can only mount a Storage Checkpoint clusterwide if the file system that the Storage Checkpoint belongs to is also mounted clusterwide. Similarly, you can only mount a Storage Checkpoint locally if the file system that the Storage Checkpoint belongs to is mounted locally.

You can unmount Storage Checkpoints using the `umount` command (see the `umount_vxfs`(1M) manual page). Storage Checkpoints can be unmounted by the mount point or pseudo device name:

```
# umount /fsvol_may_23
# umount /dev/vx/dsk/fsvol/vol1:may_23
/dev/vx/dsk/fsvol/vol1 /fsvol vxfs defaults 0 2
/dev/vx/dsk/fsvol/vol1:may_23 /fsvol_may_23 vxfs clone=may_23 0 0
```

**Note** You do not need to run the `fsck` utility on Storage Checkpoint pseudo devices because pseudo devices are part of the actual file system.

## Converting a Data Storage Checkpoint to a Nodata Storage Checkpoint

A nodata Storage Checkpoint does not contain actual file data. Instead, this type of Storage Checkpoint contains a collection of markers indicating the location of all the changed blocks since the Storage Checkpoint was created (see "Types of Storage Checkpoints" on page 70 for more information).

You can use either the *synchronous* or *asynchronous* method to convert a data Storage Checkpoint to a nodata Storage Checkpoint; the asynchronous method is the default method. In a synchronous conversion, `fsckptadm` waits for all files to undergo the conversion process to "nodata" status before completing the operation. In an asynchronous conversion, `fsckptadm` returns immediately and marks the Storage Checkpoint as a nodata Storage Checkpoint even though the Storage Checkpoint's data blocks are not immediately returned to the pool of free blocks in the file system. The Storage Checkpoint deallocates all of its file data blocks in the background and eventually returns them to the pool of free blocks in the file system.

If all of the older Storage Checkpoints in a file system are nodata Storage Checkpoints, use the synchronous method to convert a data Storage Checkpoint to a nodata Storage Checkpoint. If an older data Storage Checkpoint exists in the file system, use the asynchronous method to mark the Storage Checkpoint you want to convert for a delayed conversion. In this case, the actual conversion will continue to be delayed until the Storage Checkpoint becomes the oldest Storage Checkpoint in the file system, or all of the older Storage Checkpoints have been converted to nodata Storage Checkpoints.

**Note** You cannot convert a nodata Storage Checkpoint to a data Storage Checkpoint because a nodata Storage Checkpoint only keeps track of the location of block changes and does not save the content of file data blocks.



## Difference Between a Data and a Nodata Storage Checkpoint

The following example shows the difference between data Storage Checkpoints and nodata Storage Checkpoints.

▼ **To show the difference between Storage Checkpoints**

1. Create a file system and mount it on `/mnt0`:

```
# mkfs -F vxfs /dev/vx/rdsk/dg1/test0
version 6 layout
1024000 sectors, 512000 blocks of size 1024, log size 1024
blocks, largefiles supported
# mount -F vxfs /dev/vx/dsk/dg1/test0 /mnt0
```

2. Create a small file with a known content. Create a Storage Checkpoint and mount it on `/mnt0@5_30pm`:

```
# echo "hello, world" > /mnt0/file
# fsckptadm create ckpt@5_30pm /mnt0
# mkdir /mnt0@5_30pm
# mount -F vxfs -o ckpt=ckpt@5_30pm \
/dev/vx/dsk/dg1/test0:ckpt@5_30pm /mnt0@5_30pm
```

3. Examine the content of the original file and the Storage Checkpoint file:

```
# cat /mnt0/file
hello, world
# cat /mnt0@5_30pm/file
hello, world
```

4. Change the content of the original file:

```
# echo "goodbye" > /mnt0/file
```

5. Examine the content of the original file and the Storage Checkpoint file. The original file contains the latest data while the Storage Checkpoint file still contains the data at the time of the Storage Checkpoint creation:

```
# cat /mnt0/file
goodbye
# cat /mnt0@5_30pm/file
hello, world
```

6. Unmount the Storage Checkpoint, convert the Storage Checkpoint to a nodata Storage Checkpoint, and mount the Storage Checkpoint again.

```
# umount /mnt0@5_30pm
# fsckptadm -s set nodata ckpt@5_30pm /mnt0
# mount -F vxfs -o ckpt=ckpt@5_30pm \
  /dev/vx/dsk/dg1/test0:ckpt@5_30pm /mnt0@5_30pm
```

7. Examine the content of both files. The original file must contain the latest data:

```
# cat /mnt0/file
goodbye
```

You can traverse and read the directories of the nodata Storage Checkpoint; however, the files contain no data, only markers to indicate which block of the file has been changed since the Storage Checkpoint was created:

```
# ls -l /mnt0@5_30pm/file
-rw-r--r--   1 root     other 13 Jul 13 17:13 /mnt0@5_30pm/file
# cat /mnt0@5_30pm/file
cat: /mnt0@5_30pm/file: I/O error
```



## Conversion with Multiple Storage Checkpoints

The following example highlights the conversion of data Storage Checkpoints to nodata Storage Checkpoints, particularly when dealing with older Storage Checkpoints on the same file system.

### ▼ To convert Storage Checkpoints

1. Create a file system and mount it on `/mnt0`:

```
# mkfs -F vxfs /dev/vx/rdsk/dg1/test0
version 6 layout
1024000 sectors, 512000 blocks of size 1024, log size 1024 blocks
largefiles supported
# mount -F vxfs /dev/vx/dsk/dg1/test0 /mnt0
```

2. Create four data Storage Checkpoints on this file system, note the order of creation, and list them:

```
# fsckptadm create oldest /mnt0
# fsckptadm create older /mnt0
# fsckptadm create old /mnt0
# fsckptadm create latest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                = Mon 26 Jul 11:56:55 2004
  mtime                = Mon 26 Jul 11:56:55 2004
  flags                = largefiles
old:
  ctime                = Mon 26 Jul 11:56:51 2004
  mtime                = Mon 26 Jul 11:56:51 2004
  flags                = largefiles
older:
  ctime                = Mon 26 Jul 11:56:46 2004
  mtime                = Mon 26 Jul 11:56:46 2004
  flags                = largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = largefiles
```

3. Try to convert synchronously the "latest" Storage Checkpoint to a nodata Storage Checkpoint. The attempt will fail because the Storage Checkpoints older than the "latest" Storage Checkpoint are data Storage Checkpoints, namely the Storage Checkpoints "old", "older", and "oldest":

```
# fsckptadm -s set nodata latest /mnt0
UX:vxfs fsckptadm: ERROR: V-3-24632: storage checkpoint set
failed on latest. File exists (17)
```

4. You can instead convert the "latest" Storage Checkpoint to a nodata Storage Checkpoint in a delayed or asynchronous manner. If you list the Storage Checkpoints, you will see that the "latest" Storage Checkpoint is marked for conversion in the future:

```
# fsckptadm set nodata latest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                = Mon 26 Jul 11:56:55 2004
  mtime                = Mon 26 Jul 11:56:55 2004
  flags                = nodata, largefiles, delayed
old:
  ctime                = Mon 26 Jul 11:56:51 2004
  mtime                = Mon 26 Jul 11:56:51 2004
  flags                = largefiles
older:
  ctime                = Mon 26 Jul 11:56:46 2004
  mtime                = Mon 26 Jul 11:56:46 2004
  flags                = largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = largefiles
```



5. You can combine the three previous steps and create the "latest" Storage Checkpoint as a nodata Storage Checkpoint. The creation process will detect the presence of the older data Storage Checkpoints and create the "latest" Storage Checkpoint as a delayed nodata Storage Checkpoint. First remove the "latest" Storage Checkpoint:

```
# fsckptadm remove latest /mnt0
# fsckptadm list /mnt0
/mnt0
old:
  ctime                = Mon 26 Jul 11:56:51 2004
  mtime                = Mon 26 Jul 11:56:51 2004
  flags                = largefiles
older:
  ctime                = Mon 26 Jul 11:56:46 2004
  mtime                = Mon 26 Jul 11:56:46 2004
  flags                = largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = largefiles
```

Then recreate it as a nodata Storage Checkpoint:

```
# fsckptadm -n create latest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                = Mon 26 Jul 12:06:42 2004
  mtime                = Mon 26 Jul 12:06:42 2004
  flags                = nodata, largefiles, delayed
old:
  ctime                = Mon 26 Jul 11:56:51 2004
  mtime                = Mon 26 Jul 11:56:51 2004
  flags                = largefiles
older:
  ctime                = Mon 26 Jul 11:56:46 2004
  mtime                = Mon 26 Jul 11:56:46 2004
  flags                = largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = largefiles
```

6. You can synchronously convert the "oldest" Storage Checkpoint to a nodata Storage Checkpoint because no older Storage Checkpoints exist that contain data in the file system:

```
# fsckptadm -s set nodata oldest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                = Mon 26 Jul 12:06:42 2004
  mtime                = Mon 26 Jul 12:06:42 2004
  flags                = nodata, largefiles, delayed
old:
  ctime                = Mon 26 Jul 11:56:51 2004
  mtime                = Mon 26 Jul 11:56:51 2004
  flags                = largefiles
older:
  ctime                = Mon 26 Jul 11:56:46 2004
  mtime                = Mon 26 Jul 11:56:46 2004
  flags                = largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = nodata, largefiles
```

7. Remove the "older" and "old" Storage Checkpoints. After you remove the "older" and "old" Storage Checkpoints, the "latest" Storage Checkpoint is automatically converted to a nodata Storage Checkpoint because the only remaining older Storage Checkpoint ("oldest") is already a nodata Storage Checkpoint:

```
# fsckptadm remove older /mnt0
# fsckptadm remove old /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                = Mon 26 Jul 12:06:42 2004
  mtime                = Mon 26 Jul 12:06:42 2004
  flags                = nodata, largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = nodata, largefiles
```



# Space Management Considerations

Several operations, such as removing or overwriting a file, can fail when a file system containing Storage Checkpoints runs out of space. Usually these operations do not fail because of insufficient space on the file system, but these operations on a file system containing Storage Checkpoints can cause a data block copy that, in turn, may require extent allocation. If the system cannot allocate sufficient space, the operation will fail.

Database applications usually preallocate storage for their files and may not expect a write operation to fail. If a file system runs out of space, the kernel automatically removes Storage Checkpoints and attempts to complete the write operation after sufficient space becomes available. The kernel removes Storage Checkpoints to prevent commands, such as `rm` (see the `rm`(1) manual page), from failing under an out-of-space (ENOSPC) condition.

The kernel will follow these policies when automatically removing Storage Checkpoints:

- Remove as few Storage Checkpoints as possible to complete the operation.
- Never select a non-removable Storage Checkpoint.
- Select a nodata Storage Checkpoint only when data Storage Checkpoints no longer exist.
- Remove the oldest Storage Checkpoint first.

# File System Restore From Storage Checkpoints

Mountable data Storage Checkpoints on a consistent and undamaged file system can be used by backup and restore applications to restore either individual files or an entire file system. Restoration from Storage Checkpoints can also help recover incorrectly modified files, but typically cannot recover from hardware damage or other file system integrity problems.

**Note** For hardware or other integrity problems, Storage Checkpoints must be supplemented by backups from other media.

Files can be restored by copying the entire file from a mounted Storage Checkpoint back to the primary fileset. To restore an entire file system, you can designate a mountable data Storage Checkpoint as the primary fileset using the `fsckpt_restore` command (see the `fsckpt_restore`(1M) manual page). When using the `fsckpt_restore` command to restore a file system from a Storage Checkpoint, all changes made to that file system after that Storage Checkpoint's creation date are permanently lost. The only Storage Checkpoints and data preserved are those that were created at the same time, or before, the selected Storage Checkpoint's creation. The file system cannot be mounted when `fsckpt_restore` is invoked.

**Note** Files can be restored very efficiently by applications using the `fsckpt_fbmap`(3) library function to restore only modified portions of a files data.

## Example of Restoring a File From a Storage Checkpoint

The following example restores a file, `MyFile.txt`, which resides in your home directory, from the Storage Checkpoint "CKPT1" to the device `/dev/vx/dsk/vol-01`. The mount point for the device is `/home`.

### ▼ To restore a file from a Storage Checkpoint

1. Create the Storage Checkpoint `CKPT1` of `/home`.

```
$ fckptadm create CKPT1 /home
```

2. Mount Storage Checkpoint `CKPT1` on the directory `/home/checkpoints/mar_4`.

```
$ mount -F vxfs -o ckpt=CKPT1 /dev/vx/dsk/dg1/vol-01:CKPT1 \
 /home/checkpoints/mar_4
```

3. Delete the file `MyFile.txt` from your home directory.

```
$ cd /home/users/me
$ rm MyFile.txt
```



4. Go to the `/home/checkpoints/mar_4/users/me` directory, which contains the image of your home directory.

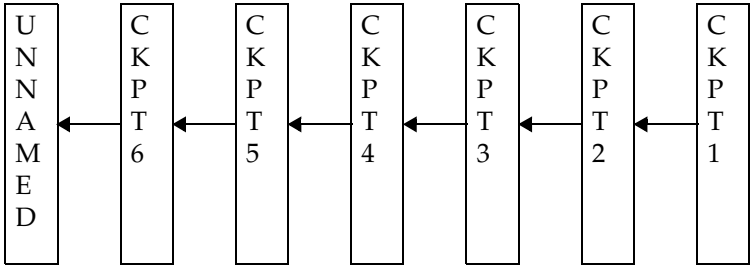
```
$ cd /home/checkpoints/mar_4/users/me
$ ls -l
-rw-r--r--   1  me   staff   14910   Mar 4 17:09 MyFile.txt
```

5. Copy the file `MyFile.txt` to your home directory.

```
$ cp MyFile.txt /home/users/me
$ cd /home/users/me
$ ls -l
-rw-r--r--   1  me   staff   14910   Mar 4 18:21 MyFile.txt
```

## Example of Restoring a File System From a Storage Checkpoint

The following example restores a file system from the Storage Checkpoint "CKPT3." The filesets listed before the restoration show an unnamed root fileset and six Storage Checkpoints.



### ▼ To restore a file system from a Storage Checkpoint

1. Run the `fsckpt_restore` command:

```
# fsckpt_restore -l /dev/vx/dsk/dg1/vol2
/dev/vx/dsk/dg1/vol2:

UNNAMED:
   ctime        = Thu 08 May 2004 06:28:26 PM PST
   mtime        = Thu 08 May 2004 06:28:26 PM PST
   flags        = largefiles, file system root

CKPT6:
   ctime        = Thu 08 May 2004 06:28:35 PM PST
   mtime        = Thu 08 May 2004 06:28:35 PM PST
   flags        = largefiles

CKPT5:
   ctime        = Thu 08 May 2004 06:28:34 PM PST
   mtime        = Thu 08 May 2004 06:28:34 PM PST
   flags        = largefiles, nomount

CKPT4:
   ctime        = Thu 08 May 2004 06:28:33 PM PST
   mtime        = Thu 08 May 2004 06:28:33 PM PST
   flags        = largefiles

CKPT3:
   ctime        = Thu 08 May 2004 06:28:36 PM PST
   mtime        = Thu 08 May 2004 06:28:36 PM PST
   flags        = largefiles

CKPT2:
   ctime        = Thu 08 May 2004 06:28:30 PM PST
   mtime        = Thu 08 May 2004 06:28:30 PM PST
   flags        = largefiles

CKPT1:
   ctime        = Thu 08 May 2004 06:28:29 PM PST
   mtime        = Thu 08 May 2004 06:28:29 PM PST
   flags        = nodata, largefiles
```



2. In this example, select the Storage Checkpoint "CKPT3" as the new root fileset:

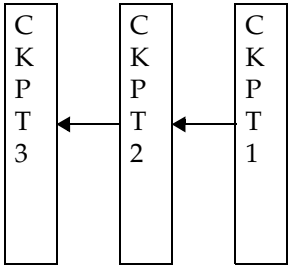
```
Select Storage Checkpoint for restore operation
  or <Control/D> (EOF) to exit
  or <Return> to list Storage Checkpoints: CKPT3
CKPT3:
   ctime        = Thu 08 May 2004 06:28:31 PM PST
   mtime        = Thu 08 May 2004 06:28:36 PM PST
   flags        = largefiles

UX:vxfs fsckpt_restore: WARNING: V-3-24640: Any file system
changes or Storage Checkpoints made after
Thu 08 May 2004 06:28:31 PM PST will be lost.
```

3. Enter "**y**" to restore the file system from CKPT3:

```
Restore the file system from Storage Checkpoint CKPT3 ? (ynq) y
(Yes)
UX:vxfs fsckpt_restore: INFO: V-3-23760: File system restored
from CKPT3
```

If the filesets are listed at this point, it shows that the former UNNAMED root fileset and CKPT6, CKPT5, and CKPT4 were removed, and that CKPT3 is now the primary fileset. CKPT3 is now the fileset that will be mounted by default.



4. Run the `fsckpt_restore` command:

```
# fsckpt_restore -l /dev/vx/dsk/dg1/vol2
/dev/vx/dsk/dg1/vol2:

CKPT3:
   ctime        = Thu 08 May 2004 06:28:31 PM PST
   mtime        = Thu 08 May 2004 06:28:36 PM PST
   flags        = largefiles, file system root

CKPT2:
   ctime        = Thu 08 May 2004 06:28:30 PM PST
   mtime        = Thu 08 May 2004 06:28:30 PM PST
   flags        = largefiles

CKPT1:
   ctime        = Thu 08 May 2004 06:28:29 PM PST
   mtime        = Thu 08 May 2004 06:28:29 PM PST
   flags        = nodata, largefiles

Select Storage Checkpoint for restore operation
  or <Control/D> (EOF) to exit
  or <Return> to list Storage Checkpoints:
```



# Storage Checkpoint Quotas

VxFS provides options to the `fsckptadm` command interface to administer Storage Checkpoint quotas. Storage Checkpoint quotas set limits on the number of blocks used by a primary fileset and all of its related Storage Checkpoints.

| | |
|---|---|
| *hard limit* | An absolute limit that cannot be exceeded. If a hard limit is exceeded, all further allocations on any of the Storage Checkpoints fail, but existing Storage Checkpoints are preserved. |
| *soft limit* | Must be lower than the hard limit. If a soft limit is exceeded, no new Storage Checkpoints can be created. The number of blocks used must return below the soft limit before more Storage Checkpoints can be created. An alert and console message are generated. |

In case of a hard limit violation, two solutions are possible, enacted by specifying or not specifying the `-f` option for the `fsckptadm` utility (see the `fsckptadm`(1M) manual page):

- If the `-f` option not is specified, one or many removable Storage Checkpoints are deleted to make space for the operation to succeed. This is the default solution.
- If the `-f` option is specified, all further allocations on any of the Storage Checkpoints fail, but existing Storage Checkpoints are preserved.

**Note** Sometimes if a file is removed while it is opened by another process, the removal process is deferred until the last close. Because the removal of a file may trigger pushing data to a "downstream" Storage Checkpoint (that is, the next older Storage Checkpoint), a fileset hard limit quota violation may occur. In this scenario, the hard limit is relaxed to prevent an inode from being marked bad. This is also true for some asynchronous inode operations.

# Online Backup Using File System Snapshots 6

This chapter describes the online backup facility provided with the VERITAS File System (VxFS). The snapshot feature of VxFS can be used to create a snapshot image of a mounted file system, which becomes a duplicate read-only copy of the mounted file system. This chapter also provides a description of how to create a snapshot file system and some examples of backing up all or part of a file system using the snapshot mechanism.

The following topics are covered in this chapter:





# Snapshot File Systems

A *snapshot file system* is an exact image of a VxFS file system, referred to as the *snapped file system*, that provides a mechanism for making backups. The snapshot is a consistent view of the file system "snapped" at the point in time the snapshot is made. You can select files to back up from the snapshot (using a standard utility such as `cpio` or `cp`), or back up the entire file system image (using the `vxdump` or `fscat` utilities).

You use the `mount` command to create a snapshot file system (the `mkfs` command is not required). A snapshot file system is always read-only. A snapshot file system exists only as long as it and the snapped file system are mounted and ceases to exist when unmounted. A snapped file system cannot be unmounted until all of its snapshots are unmounted. Although it is possible to have multiple snapshots of a file system made at different times, it is not possible to make a snapshot of a snapshot.

---

**Note** A snapshot file system ceases to exist when unmounted. If mounted again, it is actually a fresh snapshot of the snapped file system.

A snapshot file system must be unmounted before its dependent snapped file system can be unmounted. Neither the `fuser` command nor the `mount` command will indicate that a snapped file system cannot be unmounted because a snapshot of it exists.

---

On cluster file systems, snapshots can be created on any node in the cluster, and backup operations can be performed from that node. The snapshot of a cluster file system is accessible only on the node where it is created, that is, the snapshot file system itself cannot be cluster mounted. See the *VERITAS SANPoint Foundation Suite Installation and Configuration Guide* for more information on creating snapshots on cluster file systems.

# Using a Snapshot File System for Backup

After a snapshot file system is created, the snapshot maintains a consistent backup of data in the snapped file system.

Backup programs (such as `cpio`) that back up a standard file system tree can be used without modification on a snapshot file system because the snapshot presents the same data as the snapped file system. Backup programs (such as `vxdump`) that access the disk structures of a file system require some modifications to handle a snapshot file system.

VxFS utilities recognize snapshot file systems and modify their behavior so that they operate the same way on snapshots as they do on standard file systems. Other backup programs that typically read the raw disk image cannot work on snapshots without altering the backup procedure.

These other backup programs can use the `fscat` command to obtain a raw image of the entire file system that is identical to an image obtainable by running a `dd` command on the disk device containing the snapped file system at the exact moment the snapshot was created. The `snapread` ioctl takes arguments similar to those of the `read` system call and returns the same results that are obtainable by performing a read on the disk device containing the snapped file system at the exact time the snapshot was created. In both cases, however, the snapshot file system provides a consistent image of the snapped file system with all activity complete—it is an instantaneous read of the entire file system. This is much different than the results that would be obtained by a `dd` or `read` command on the disk device of an active file system.

If you create a complete backup of a snapshot file system using a utility such as `vxdump` and later restore it, you must run the `fsck` command on the restored file system because the snapshot file system is consistent, but not clean. That is, the file system may have some extended inode operations to complete, but there should be no other changes. Because a snapshot file system is not writable, it cannot be fully checked, but the `fsck -n` command can be used to report any inconsistencies.

# Creating a Snapshot File System

You create a snapshot file system by using the `-o snapof=` option of the `mount` command. The `-o snapsize=` option may also be required if the device you are mounting does not identify the device size in its disk label, or if you want a size smaller than the entire device. Use the following syntax to create a snapshot file system:

```
# mount -F vxfs -o snapof=special,snapsize=snapshot_size \
  snapshot_special snapshot_mount_point
```

You must make the snapshot file system large enough to hold any blocks on the snapped file system that may be written to while the snapshot file system exists. If a snapshot runs out of blocks to hold copied data, it is disabled and further attempts to access the snapshot file system fail.

During periods of low activity (such as nights and weekends), a snapshot typically requires about two to six percent of the blocks of the snapped file system. During a period of high activity, the snapshot of a typical file system may require 15 percent of the blocks of the snapped file system. Most file systems do not turn over 15 percent of data in a single day. These approximate percentages tend to be lower for larger file systems and higher for smaller file systems. You can allocate blocks to a snapshot based on characteristics such as file system usage and duration of backups.

**Caution** Any existing data on the device used for the snapshot is overwritten.



# Making a Backup

Here are some typical examples of making a backup of a 300,000 block file system named `/home` using a snapshot file system on `/dev/vx/dsk/fsvol/vol1` with a snapshot mount point of `/backup/home`:

- To back up files changed within the last week using `cpio`:

```
# mount -F vxfs -o snapof=/home,snapsize=100000 \
/dev/vx/dsk/fsvol/vol1 /backup/home
# cd /backup
# find home -ctime -7 -depth -print | cpio -oc > /dev/rmt/0m
# umount /backup/home
```

- To do a full backup of `/home`, which exists on disk `/dev/vx/dsk/fsvol/vol1`, and use `dd` to control blocking of output onto tape device using `vxdump`:

```
# vxdump f - /dev/vx/dsk/fsvol/vol1 | dd bs=128k > /dev/rmt/0m
```

- To do a level 3 backup of `/dev/vx/dsk/fsvol/vol1` and collect those files that have changed in the current directory:

```
# vxdump 3f - /dev/vx/dsk/fsvol/vol1 | vxrestore -xf -
```

- To do a full backup of a snapshot file system:

```
# mount -F vxfs -o snapof=/home,snapsize=100000 \
/dev/vx/dsk/fsvol/vol1 /backup/home
# vxdump f - /dev/vx/dsk/fsvol/vol1 | dd bs=128k > /dev/rmt/0m
```

The `vxdump` utility ascertains whether `/dev/vx/dsk/fsvol/vol1` is a snapshot mounted as `/backup/home` and do the appropriate work to get the snapshot data through the mount point.

# Performance of Snapshot File Systems

Snapshot file systems maximize the performance of the snapshot at the expense of writes to the snapped file system. Reads from a snapshot file system typically perform at nearly the throughput rates of reads from a standard VxFS file system.

The performance of reads from the snapped file system are generally not affected. However, writes to the snapped file system, typically average two to three times as long as without a snapshot. This is because the initial write to a data block requires reading the old data, writing the data to the snapshot, and then writing the new data to the snapped file system. If there are multiple snapshots of the same snapped file system, writes are even slower. Only the initial write to a block experiences this delay, so operations such as writes to the intent log or inode updates proceed at normal speed after the initial write.

Reads from the snapshot file system are impacted if the snapped file system is busy because the snapshot reads are slowed by the disk I/O associated with the snapped file system.

The overall impact of the snapshot is dependent on the read to write ratio of an application and the mixing of the I/O operations. For example, a database application running an online transaction processing (OLTP) workload on a snapped file system was measured at about 15 to 20 percent slower than a file system that was not snapped.

# Differences Between Snapshots and Storage Checkpoints

While snapshots and Storage Checkpoints both create a *point-in-time* image of a file system and only the changed data blocks are updated, there are significant differences between the two technologies:

- Snapshots require a separate device for storage. Storage Checkpoints reside on the same device as the original file system.
- Snapshots are read-only. Storage Checkpoints can be read-only or read-write.
- Snapshots are transient. Storage Checkpoints are persistent.
- Snapshots cease to exist after being unmounted. Storage Checkpoints can exist and be mounted on their own
- Snapshots track changed blocks on the file system level. Storage Checkpoints track changed blocks on each file in the file system.
- Although there can be more than one snapshot of a file system, they are all based on a single, parent file system. Storage Checkpoints can be based on other Storage Checkpoints.

Storage Checkpoints also serve as the enabling technology for two other VERITAS features: *Block-Level Incremental Backups* and *Storage Rollback*, which are used extensively for backing up databases. See "Storage Checkpoints" on page 65 for more information.



# Snapshot File System Internals

The following sections describe the internal structure of a snapshot file system and how it copies changed data blocks from the original snapped file system.

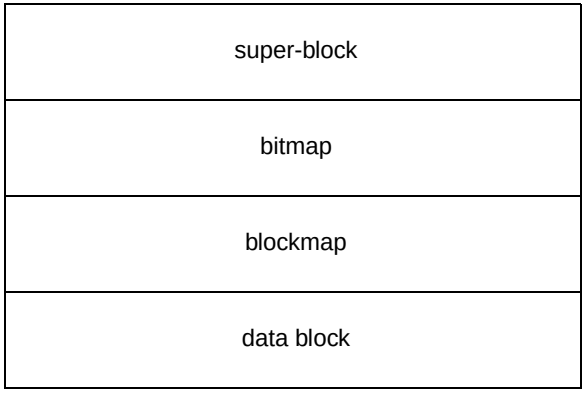
## Snapshot File System Disk Structure

A snapshot file system consists of:

- A super-block
- A bitmap
- A blockmap
- Data blocks copied from the snapped file system

The following figure shows the disk structure of a snapshot file system:

The Snapshot Disk Structure




The super-block is similar to the super-block of a standard VxFS file system, but the magic number is different and many of the fields are not applicable.

The bitmap contains one bit for every block on the snapped file system. Initially, all bitmap entries are zero. A set bit indicates that the appropriate block was copied from the snapped file system to the snapshot. In this case, the appropriate position in the blockmap references the copied block.

The blockmap contains one entry for each block on the snapped file system. Initially, all entries are zero. When a block is copied from the snapped file system to the snapshot, the appropriate entry in the blockmap is changed to contain the block number on the snapshot file system that holds the data from the snapped file system.

The data blocks are filled by data copied from the snapped file system, starting from the beginning of the data block area.

## How a Snapshot File System Works

A snapshot file system is created by mounting an empty disk slice as a snapshot of a currently mounted file system. The bitmap, blockmap and super-block are initialized and then the currently mounted file system is frozen (see "Freeze and Thaw" on page 60, for a description of the VX_FREEZE ioctl). After the file system to be snapped is frozen, the snapshot is enabled and mounted and the snapped file system is thawed. The snapshot appears as an exact image of the snapped file system at the time the snapshot was made.

Initially, the snapshot file system satisfies read requests by finding the data on the snapped file system and returning it to the requesting process. When an inode update or a write changes the data in block *n* of the snapped file system, the old data is first read and copied to the snapshot before the snapped file system is updated. The bitmap entry for block *n* is changed from 0 to 1 (indicating that the data for block *n* can be found on the snapped file system). The blockmap entry for block *n* is changed from 0 to the block number on the snapshot file system containing the old data.

A subsequent read request for block *n* on the snapshot file system will be satisfied by checking the bitmap entry for block *n* and reading the data from the indicated block on the snapshot file system, instead of from block *n* on the snapped file system. This technique is called *copy-on-write*. Subsequent writes to block *n* on the snapped file system do not result in additional copies to the snapshot file system, since the old data only needs to be saved once.

All updates to the snapped file system for inodes, directories, data in files, extent maps, and so forth, are handled in this fashion so that the snapshot can present a consistent view of all file system structures on the snapped file system for the time when the snapshot was created. As data blocks are changed on the snapped file system, the snapshot gradually fills with data copied from the snapped file system.



The amount of disk space required for the snapshot depends on the rate of change of the snapped file system and the amount of time the snapshot is maintained. In the worst case, the snapped file system is completely full and every file is removed and rewritten. The snapshot file system would need enough blocks to hold a copy of every block on the snapped file system, plus additional blocks for the data structures that make up the snapshot file system. This is approximately 101 percent of the size of the snapped file system. Normally, most file systems do not undergo changes at this extreme rate. During periods of low activity, the snapshot should only require two to six percent of the blocks of the snapped file system. During periods of high activity, the snapshot might require 15 percent of the blocks of the snapped file system. These percentages tend to be lower for larger file systems and higher for smaller ones.

**Caution** If a snapshot file system runs out of space for changed data blocks, it is disabled and all further access to it fails. This does not affect the snapped file system.

# Quotas 7

The VERITAS File System (VxFS) supports user quotas. The quota system limits the use of two principal resources of a file system: files and data blocks. For each of these resources, you can assign quotas to individual users to limit their usage.

The following topics are covered in this chapter:





# Quota Limits

You can set limits for individual users to file and data block usage on a file system. You can set two kinds of limits for each of the two resources:

- The *hard limit* is an absolute limit that cannot be exceeded under any circumstances.
- The *soft limit*, which must be lower than the hard limit, can be exceeded, but only for a limited time. The time limit can be configured on a per-file system basis only. The VxFS default limit is seven days.

A typical use of soft limits is when a user must run an application that could generate large temporary files. In this case, you can allow the user to exceed the quota limit for a limited time. No allocations are allowed after the expiration of the time limit. Use the `edquota` command to set limits (see "Using Quotas" on page 99 for an example).

Although file and data block limits can be set individually for each user , the time limits apply to the file system as a whole. The quota limit information is associated with user IDs and is stored in a user  quota file (see "Quota Files on VxFS" below).

The quota soft limit can be exceeded when VxFS preallocates space to a file. See "Attribute Specifics" on page 48 for information on extent allocation policies.

Quota limits cannot exceed two terabytes on a Version 5 disk layout.

# Quota Files on VxFS

A quotas file (named `quotas`) must exist in the root directory of a file system for any of the quota commands to work. The files in the file system's mount point are referred to as the *external* quotas file. VxFS also maintains an *internal* quotas file for its own use.

The quota administration commands read and write to the external `quotas` file to obtain or change usage limits. VxFS uses the internal file to maintain counts of data blocks and inodes used by each user. When quotas are turned on, the quota limits are copied from the external `quotas` file into the internal `quotas` file. While quotas are on, all the changes in the usage information and changes to quotas are registered in the internal `quotas` file. When quotas are turned off, the contents of the internal `quotas` file are copied into the external `quotas` file so that all data between the two files is synchronized.

# Quota Commands

Most of the quotas commands in VxFS are similar to BSD quotas commands. In general, quota administration for VxFS is performed using commands similar to HFS quota commands. The VxFS `mount` command supports a special mount option (`–o quota`), that can be used to turn on quotas at mount time.

**Note** For additional information on the quota commands, see the corresponding manual pages. When VxFS file systems are exported via NFS, the VxFS quota commands on the NFS client cannot query or edit quotas. You can use the VxFS quota commands on the server to query or edit quotas.

# Using Quotas

This section shows usage examples of the VxFS quota commands.

## quotaon

To use the quota functionality on a file system, quotas must be turned on. You can turn them on at mount time or after a file system is mounted.

**Note** Before turning on quotas, the root directory of the file system must contain a file for user quotas named `quotas` owned by `root`.

To turn on quotas for a VxFS file system, enter:

```
# quotaon /mount_point
```

## mount

You can also turn on quotas for a file system at mount time by specifying the `–o quota` option to the `mount` command:

```
# mount –F vxfs –o quota special /mount_point
```



## edquota

You can set up user quotas using the `edquota` command. You must have superuser privileges to edit quotas:

```
# edquota username
```

`edquota` creates a temporary file for the given user; this file contains on–disk quotas for each mounted file system that has a quotas file. It is not necessary that quotas be turned on for `edquota` to work. However, the quota limits are applicable only after quotas are turned on for a given file system.

The soft and hard limits can be modified or assigned values. For any user, usage can never exceed the hard limit after quotas are turned on. Time limits can be modified using the command:

```
# edquota –t
```

Modified time limits apply to the entire file system and cannot be set selectively for each user.

## quota

Use the `quota` command to view a user's disk quotas and usage on VxFS file systems:

```
# quota –v username
```

This displays the user's quotas and disk usage on all mounted VxFS file systems where the `quotas` file exists.

## quot

Use the `quot` command to display the number of blocks owned by each user in a file system. The following command displays the number of files and the space owned by each user:

```
# quot –f filesystem
```

## quotaoff

To turn off quotas for a mounted file system, enter:

```
# quotaoff /mount_point
```

# File Change Log 8

**Caution** File Change Log is currently not officially supported, and VERITAS strongly cautions against using it in a production environment. Although FCL is not 100% complete, it is functional and can be used to begin developing new applications. File Change Log will be fully operational in the next VERITAS File System maintenance release.

The VxFS File Change Log (FCL) tracks changes to files and directories in a file system. Applications that can make use of FCL are those that are typically required to scan an entire file system to discover changes since the last scan, such as backup utilities, webcrawlers, search engines, and replication programs.

The FCL records file system changes such as creates, links, unlinks, renaming, data appended, data overwritten, data truncated, extended attribute modifications, holes punched, and miscellaneous file property updates.

**Note** FCL records only that data has changed, not the actual data. It is the responsibility of the application to examine the files that have changed data to determine which data has changed.

The FCL feature is not available on file systems created with the `nolargefiles` option.

FCL functionality is a separately licensable feature. See the *VERTIAS Storage Foundation Release Notes* for more information.

## File Change Log File

FCL stores changes in a sparse file in the file system namespace. The FCL log file is always located in `mount_point/lost+found/changelog`. The FCL file behaves like a regular file, but some operations are prohibited. The standard system calls `open`(2), `lseek`(2), `read`(2) and `close`(2) can access the data in the FCL, while the `write`(2), `mmap`(2) and `rename`(2) calls are not allowed.



The FCL log file contains both the information about the FCL (stored in the FCL superblock), and the changes to files and directories in the file system, stored as *FCL records*. Details on the structure and semantics of the FCL superblock and FCL records, and the types of changes tracked by the FCL, are located in the header file `/opt/VRTS/include/sys/fs/fcl.h` (see "File Change Log Programmatic Interface" on page 103).

# File Change Log Administrative Interface

The FCL can be set up and tuned through the VxFS administrative commands `fcladm`(1M) and `vxtunefs`(1M). The FCL tunable parameters are:

fcl_keeptime — Specifies the duration in seconds that FCL records stay in the FCL file before they can be purged. The first records to be purged are the oldest ones, which are located at the beginning of the file. Additionally, records at the beginning of the file can be purged if allocation to the FCL file exceeds *fcl_maxalloc* bytes. The default value is 0. Note that *fcl_keeptime* takes precedence over *fcl_maxalloc*. No hole is punched if the FCL file exceeds *fcl_maxalloc* bytes but the life of the oldest record has not reached *fcl_keeptime* seconds.

fcl_maxalloc — Specifies the maximum number of spaces in bytes to be allocated to the FCL file. When the space allocated exceeds *fcl_maxalloc*, a hole is punched at the beginning of the file. As a result, records are purged and the first valid offset (*fc_foff*) is updated. The minimum value of *fcl_maxalloc* is 4MB. The default value is *fs_size*/33.

fcl_winterval — Specifies the time in seconds that must elapse before the FCL records an overwrite, extending write, or a truncate. This helps to reduce the number of repetitive records in the FCL. *fcl_winterval* time-out is per inode. If an inode happens to go out of cache and returns, its write interval is reset. As a result, there could be more than one write record for that file in the same write interval. The default value is 3600 seconds.

Either or both `fcl_maxalloc`/`fcl_keeptime` must be set to activate the FCL. The following are examples of using the FCL administration command.

To activate the FCL for a mounted file system, enter:

```
# fcladm on mount_point
```

To deactivate the FCL for a mounted file system, enter:

```
# fcladm off mount_point
```

To remove the FCL file for a mounted file system (the FCL must be `OFF` before it can be removed), enter:

```
# fcladm rm mount_point
```

To obtain the current FCL state for a mounted file system, enter:

```
# fcladm state mount_point
```

Print the on-disk FCL super-block in text format to obtain information about the FCL by using offset 0. Because the FCL on-disk super-block occupies the first block of the FCL file, the first and last valid offsets into the FCL file can be determined by reading the FCL super-block and checking the `fc_foff` field. Enter:

```
# fcladm print 0 mount_point
```

To print the contents of the FCL in text format (the offset used must be 32-byte aligned), enter:

```
# fcladm print offset mount_point
```

# File Change Log Programmatic Interface

The standard system calls `open`(2), `lseek`(2), `read`(2) and `close`(2) can be used on the FCL file at `mount_point/lost+found/changelog`. Only one programmatic interface is exposed through `libvxfsutil`, the `vxfs_fcl_sync` API (see the `vxfs_fcl_sync`(3) manual page). The prototype is available at `/opt/VRTSfssdk/4.1.00.0/include/vxfsutil.h`.

The following sample code fragment reads the FCL superblock, checks that the state of the FCL is VX_FCLS_ON, issues a call to `vxfs_fcl_sync` to obtain a finishing offset to read to, determines the first valid offset in the FCL file, then reads the entries in 8K chunks from this offset. The section `process fcl entries` is what an application developer must supply to process the entries in the FCL.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/fcntl.h>
#include <errno.h>
#include <fcl.h>
#include <vxfsutil.h>

#define FCL_READSZ 8192

char* fclname = "/mnt/lost+found/changelog";
```



```
int
read_fcl(fclname)
     char* fclname;
{
     struct fcl_sb fclsb;
     uint64_t off, lastoff;
     size_t size;
     char buf[FCL_READSZ], *bufp = buf;
     int fd;
     int err = 0;

     if ((fd = open(fclname, O_RDONLY)) < 0) {
               return ENOENT;
     }
     if ((off = lseek(fd, 0, SEEK_SET)) != 0) {
               close(fd);
               return EIO;
     }
     size = read(fd, &fclsb, sizeof (struct fcl_sb));
     if (size < 0) {
               close(fd);
               return EIO;
     }
     if (fclsb.fc_state == VX_FCLS_OFF) {
               close(fd);
               return 0;
     }
     if (err = vxfs_fcl_sync(fclname, &lastoff)) {
               close(fd);
               return err;
     }
     if ((off = lseek(fd, fclsb.fc_foff, SEEK_SET)) !=
          fclsb.fc_foff) {
               close(fd);
               return EIO;
     }
     while (off < lastoff) {
               if ((size = read(fd, bufp, FCL_READSZ)) <= 0) {
                         close(fd);
                         return errno;
               }
               /* process fcl entries */
               off += size;
     }
     close(fd);
     return 0;
```

```
}
```

# Reverse Path Name Lookup

The reverse path name lookup feature obtains the full path name of a file or directory from the inode number of that file or directory. The inode number is provided as an argument to the `vxlsino` administrative command, or the `vxfs_inotopath` application programming interface library function.

The reverse path name lookup feature can be useful for a variety of applications, such as for clients of the VxFS file change log feature, in backup and restore utilities, and for replication products. Typically, these applications store information by inode numbers because a path name for a file or directory can be very long, thus the need for an easy method of obtaining a path name.

An inode is a unique identification number for each file in a file system. An inode contains the data and metadata associated with that file, but does not include the file name to which the inode corresponds. It is therefore relatively difficult to determine the name of a file from an inode number. The `ncheck` command provides a mechanism for obtaining a file name from an inode identifier by scanning each directory in the file system, but this process can take a long period of time. The VxFS reverse path name lookup feature obtains path names relatively quickly.

**Note** Because symbolic links do not constitute a path to the file, the reverse path name lookup feature cannot track symbolic links to files.

Because of the possibility of errors with processes renaming or unlinking and creating new files, it is advisable to perform a lookup (or open) with the path name and verify that the inode number matches the path names obtained.

See the `vxlsino`(1M), `vxfs_inotopath_gen`(3) and `vxfs_inotopath`(3) manual pages for more information.



# Multi-Volume File Systems 9

VxFS provides support for multi-volume file systems when used in conjunction with the VERITAS Volume Manager. Using multi-volume support (MVS), a single file system can be created over multiple volumes, each volume having its own properties. For example, it is possible to place metadata on mirrored storage while placing file data on better performing volume types such as RAID5.

The MVS feature also allows file systems to reside on different classes of devices, so that a file system can be supported from both inexpensive disks and from expensive arrays. Using the MVS administrative interface, you can control which data goes on which volume types.

Topics in this chapter include:





# Features Implemented Using MVS

Features that can be implemented using multi-volume support include the following:

- Controlling where files are stored can be selected at multiple levels so that specific files or file hierarchies can be assigned to different volumes. This functionality is available in the VERITAS File System Quality of Storage Service (QoSS) feature (see "Quality of Storage Service" on page 115).
- Placing the VxFS intent log on its own volume to minimize disk head movement and thereby increase performance. This functionality can be used to migrate from the VERITAS QuickLog™ feature.
- Separating Storage Checkpoints so that data allocated to a Storage Checkpoint is isolated from the rest of the file system.
- Separating metadata from file data.
- Encapsulating volumes so that a volume appears in the file system as a file. This is particularly useful for databases that are running on raw volumes.

To use the multi-volume file system features, the VERITAS Volume Manager must be installed and the Volume Set feature must be accessible.

# Volume Sets

The VERITAS Volume Manager exports a feature called *volume sets*. Unlike the traditional Volume Manager volume, which can be used for raw I/O access or to contain a file system, a volume set is a container for multiple different volumes. Each volume can have its own geometry.

The Volume Manager `vxvset` command is used to create and manage volume sets. For example, the following command creates a new volume set from an existing volume named `vol1`:

```
# vxvset make myvset vol1
```

The following commands create two new volumes and add them to the volume set:

```
# vxassist make vol2 50m
# vxassist make vol3 50m
# vxvset addvol myvset vol2
# vxvset addvol myvset vol3
```

The following command lists the component volumes of the previously created volume set:

```
# vxvset list myvset
VOLUME       INDEX      LENGTH        STATE      CONTEXT
vol1           0        20480         ACTIVE        -
vol2           1        102400        ACTIVE        -
vol3           2        102400        ACTIVE        -
```

When a volume set is created, the volumes contained by the volume set are removed from the namespace and are instead accessed through the volume set name, as shown by the output of the `ls` command:

```
# ls -l /dev/vx/rdsk/rootdg/myvset
1 root  root    108,70009 May 21 15:37 /dev/vx/rdsk/rootdg/myvset
```

However, when a volume is added to the volume set, it is no longer visible in the namespace, as shown in the following example:

```
# vxassist make vol4 50m
# ls -l /dev/vx/rdsk/rootdg/vol4
crw-- 1 root root 108,70012 May 21 15:43 /dev/vx/rdsk/rootdg/vol4
# vxvset addvol myvset vol4
# ls -l /dev/vx/rdsk/rootdg/vol4
/dev/vx/rdsk/rootdg/vol4: No such file or directory
```

Volume sets cannot be empty, so when the last entry is removed, the volume set itself is removed.



# Creating MVS File Systems

After a volume set is created, creating a VxFS file system is the same as creating a file system on a raw device or volume. You must specify the volume set name as an argument to `mkfs` as shown in the following example:

```
# mkfs -F vxfs /dev/vx/rdsk/rootdg/myvset
version 6 layout
327680 sectors, 163840 blocks of size 1024, log size 1024 blocks
largefiles supported
```

**Note** MVS is available only on file systems using disk layout Version 6. See Appendix C, "Disk Layout" for more information about disk layout versions.

After the file system is created, VxFS allocates space from the different volumes within the volume set. You can list the component volumes of the volume set using of the `fsvoladm` command:

```
# mount -F vxfs /dev/vx/dsk/rootdg/myvset /mnt1
# fsvoladm list /mnt1
devid    size          used          avail         name
0        10240         1280          8960          vol1
1        51200         16            51184         vol2
2        51200         16            51184         vol3
3        51200         16            51184         vol4
```

To add a new volume, first add the volume to the volume set, then add it to the file system:

```
# vxassist make vol5 50m
# vxvset addvol myvset vol5
# fsvoladm add /mnt1 vol5 50m
# fsvoladm list /mnt1
devid    size          used          avail         name
0        10240         1300          8940          vol1
1        51200         16            51184         vol2
2        51200         16            51184         vol3
3        51200         16            51184         vol4
4        51200         16            51184         vol5
```

A volume must be empty before you can remove it from the file system. With the exception of volume additions and deletions, file system commands operate the same on volumes within a volume set.

# Allocation Policies

To make full use of multi-volume support features, VxFS provides support for allocation policies that allow files or groups of files to be assigned to specified volumes within the volume set.

A policy specifies a list of volumes and the order in which to attempt allocations. A policy can be assigned to a file, to a file system, or to a Storage Checkpoint created from a file system. When policies are assigned to objects in the file system, you must specify how the policy maps to both metadata and file data. For example, if a policy is assigned to a single file, the file system must know where to place both the file data and metadata. If no policies are specified, the file system places data randomly.

The following example shows how allocation policies work. Assume that there are two volumes from different classes of storage:

```
# vxvset -g rootdg list myvset
VOLUME       INDEX      LENGTH        STATE      CONTEXT
vol1           0        102400        ACTIVE        -
vol2           1        102400        ACTIVE        -
```

Create a file system on the `myvset` volume set and mount it:

```
# mkfs -F vxfs /dev/vx/rdsk/rootdg/myvset
version 6 layout
204800 sectors, 102400 blocks of size 1024, log size 1024 blocks
largefiles supported

# mount -F vxfs /dev/vx/dsk/rootdg/myvset /mnt1
```

Use the following `fsapadm` commands to define two allocation policies called "datapolicy" and "metadatapolicy" to refer to the `vol1` and `vol2` volumes:

```
# fsapadm define /mnt1 datapolicy vol1
# fsapadm define /mnt1 metadatapolicy vol2
```

Assign these policies at the file system level. The data policy must be specified before the metadata policy:

```
# fsapadm assignfs /mnt1 datapolicy metadatapolicy
# fsvoladm list /mnt1
devid   size          used           avail          name
0       51200         1250           49950          vol1
1       51200         16             51184          vol2
```

The assignment of the policies on a file system-wide basis ensures that any metadata allocated is stored on the device with the policy "metadatapolicy" (`vol2`) and all user data is be stored on `vol1` with the associated "datapolicy" policy.



The effect of creating a number of files is shown in the following script:

```
i=1
while [ $i -lt 1000 ]
do
   dd if=/dev/zero of=/mnt1/$i bs=65536 count=1
   i=‘expr $i + 1‘
done
```

Before the script completes, it runs out of space even though space is still available on the `vol2` volume:

```
# fsvoladm list /mnt1
devid   size          used          avail          name
0       51200         51200         0              vol1
1       51200         221           50979          vol2
```

To allocate user data from the `vol1` volume and then use `vol2` if space runs out, assign the allocation policy as follows:

```
# fsapadm define /mnt1 datapolicy vol1 vol2
```

You must have system administrator privileges to create, remove, change policies, or set file system or Storage Checkpoint level policies. Users can assign a pre-existing policy to their files if the policy allows that. Policies can be inherited for new files.

# Volume Encapsulation

Multi-volume support enables the ability to encapsulate an existing raw volume and make the volume contents appear as a file in the file system. There are two steps required to achieve this:

- Add the volume to an existing volume set.
- Add the volume to the file system using `fsvoladm`.

As an example, assume that the following volume set and new volume exist:

```
# vxvset list myvset
VOLUME       INDEX      LENGTH        STATE      CONTEXT
vol1           0        102400        ACTIVE        -
vol2           1        102400        ACTIVE        -
```

The volume set has two volumes. Create a third volume as part of the `passwd` file and write it to the volume. This is to demonstrate how the volume can be accessed as a file as shown later:

```
# vxassist make dbvol 100m
# dd if=/etc/passwd of=/dev/vx/rdsk/rootdg/dbvol count=1
1+0 records in
1+0 records out
```

Create a file system on the volume set and mount it. The new volume is added to the volume set:

```
# mkfs –F vxfs /dev/vx/rdsk/rootdg/myvset
version 6 layout
204800 sectors, 102400 blocks of size 1024, log size 1024 blocks
largefiles not supported

# mount –F vxfs /dev/vx/dsk/rootdg/myvset /mnt1
# vxvset addvol myvset dbvol
```

The final step is to call `fsvoladm` to perform the encapsulation:

```
# fsvoladm encapsulate /mnt1/dbfile dbvol 100m
# ls -l /mnt1/dbfile
-rw------- 1 root other 104857600 May 22 11:30 /mnt1/dbfile

# head -2 /mnt1/dbfile
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
```

Note that the `passwd` file that was written to the raw volume is now visible in the new file.

---

**Note** If the encapsulated file is changed in any way, for example, extended, truncated, or moved with an allocation policy or resized volume, or the volume is encapsulated with a bias, the file cannot be de-encapsulated.

---



# Quality of Storage Service 10

The VERITAS File System allows the creation of a file system that spans multiple volumes, known as a *multi-volume file system*. The component volumes compose a volume set. See "Multi-Volume File Systems" on page 107 for more information. On top of this basic capability is a set of services known as Quality of Storage Service (QoSS).

Using different volumes can enhance performance for applications that access specific types of files by managing where the types of files are located. For example, if you have files that are infrequently accessed, you can place them on slower media that is relatively inexpensive, but which has a slower access time.

You can configure policies that automatically relocate files from one volume to another, or you can relocate files by running file relocation commands. You can think of the QoSS as having two parts:

- *Relocation policies*. Policies that you configure to determine which files to move to different component volume.
- *File relocation*. The *fssweep* utility recursively examines a multi-volume file system searching for files that match configured relocation policies. After searching a file system, `fssweep` passes information about files that match policies to the *fsmove* utility. The `fsmove` utility reads a list of file names and locations that `fssweep` has created and relocates files based on that relocation list. These utilities are contained in the `VRTSfspro` package.

**Note** VERITAS Quality of Storage Service is a licensed feature. You must purchase a separate license key for QoSS to operate. See the *VERITAS Storage Foundation Release Notes* for current product information.



# How File Relocation Works

File relocation is the process of searching a file system to select files that a relocation policy determines can be relocated, then relocating the selected files.

The volume on which files are originally located is referred to as a *source component volume*, and the volume to which files are relocated is referred to as a *destination component volume*.

To use file relocation, you configure policies to determine the files to select for relocation. The `fssweep` utility traverses a directory structure applying selection rules from the configuration file, selects eligible files, then relocates the files from the source component volumes to destination component volumes.

# Configuring Relocation Policies

Relocation policies define the files to move and the physical devices on which to locate the files. Policies are based on the following criteria:

- A file's age, which is the time since it was last accessed or last modified
- A file's size
- A file's location in a directory structure
- A file name pattern, such as the similarly named output files of an application or a common extension such as `*.gif`. All shell wild cards may be specified.
- A file's source component volume
- A file's destination component volume

If a file system has multiple component volumes, you can configure relocation policies. You configure policies using the VERITAS Enterprise Administrator (VEA) graphical user interface or using the command-line interface. See the *VERITAS Enterprise Administrator Getting Started* guide and VEA online help for information on configuring QoSS using the VEA GUI.

**Note** The `VRTSfppm` package must be installed for the VERITAS Quality of Storage Service feature to operate.

## Running fssweep

The `fssweep` command uses the following syntax:

```
fssweep [–n count –s size –t time] [–p policy] [–r] [filesystem]
```

The `fssweep` utility traverses the directory structure of one or more file systems, selecting files that meet criteria specified in relocation policies. The `fssweep` utility writes either a list of selected file names and destination volumes, or a statistical summary of selected files, to standard output.

The list of volumes that can be used for file relocation is based on the destination component volumes defined by the QoSS configuration. By default, `fssweep` writes the files and destination volumes to standard output.

Command options let you specify criteria to select files to relocate.

The fssweep Command Options

| Option | Description |
|---|---|
| `-n` *count* | Sets a limit for the maximum number of files for the `fsmove` utility to relocate. |
| `-p` *policy* | Evaluates files based on the specified *policy* only. If not specified, `fssweep` evaluates files based on all configured policies. |
| `-r` | Reports a statistical summary of files that can be relocated to standard output without actually relocating any files. The report is the same as the `fsmove` statistical summary. |
| `-s` *size* | Sets a limit for the cumulative size, in kilobytes, of all files for the `fsmove` utility to relocate. |
| `-t` *time* | Sets a limit for the cumulative time, in seconds, that `fssweep` runs. |
| *filesystem* | Evaluates files only in the specified *filesystem*. If you do not specify *filesystem*, `fssweep` evaluates all multi-volume file systems defined in the configuration file, starting at their mount points. |



## Running fsmove

The `fsmove` command uses the following syntax:

```
fsmove [destination path1 [path2,path3,...]
```

The `fsmove` utility relocates files to a destination component volume. The destination component volume and the files or directories are specified by the `fssweep` utility. The `fsmove` utility reads standard input and moves all files it reads from the output stream.

If you specify an optional destination and one or more path names, the `fsmove` utility moves the specified file or files to the destination component volume and exits. If you specify a directory, `fsmove` recursively searches through all files and subdirectories in the specified directory and moves them all to the specified destination component volume.

The fsmove Command Options

| Option | Description |
|---|---|
| *destination* | Specifies a destination component volume for the specified file or directory. |
| *path* | Specifies directory or file names to relocate to a destination volume. |

After the command completes processing, it writes a statistical summary with the following headings to standard output:

*source_volume destination_volume number_of_files_relocated size_of_all_files_relocated*

| | |
|---|---|
| *source_volume* | The source component volume where the file originally resided. |
| *destination_volume* | The destination component volume to which to relocate the file. |
| *number_of_files_relocated* | The cumulative number of all files relocated. |
| *size_of_all_files_relocated* | The cumulative size of all files relocated. |

### Scheduling Example

You can use `crontab` to schedule file selection and relocation at specified intervals.

To schedule automatic relocations, run `crontab` to include an example such as the following in a `crontab` file. The example `crontab` file entry selects files eligible for relocation and moves them to destination component volumes. The relocation process occurs once every three days at 12:30 A.M.

```
30 0 1,4,7,10,13,16,19,22,25,28 * * /opt/VRTS/bin/fssweep |
/opt/VRTS/bin/fsmove
```

The first two fields specify the time to run the job. `30 0` is 12:30 A.M.

The next field defines how often to run the process. The list of digits is the days of the month on which to run. The first asterisk (*) specifies that the process runs every month. The second asterisk specifies that days of the week are not used as criteria for when to run. See the `crontab`(5) manual page for more information.

The last two fields specify the commands to run, which are the `fssweep` and `fsmove` commands.

## Customizing QoSS

The following information is not essential to QoSS daily operation and management. This section describes the relationship between the `fssweep` and `fsmove` utilities to allocation policies and how files and volumes are selected for relocation.

### Mapping Relocation Policies to Allocation Policies

The `fssweep` and `fsmove` utilities use relocation policies to relocate existing files. Relocation uses the file system's allocation policies to determine the current location of each file and to move files to new locations. Allocation policies also control newly created files. You can manage allocation policies using the `fsapadm` utility (see the `fsapadm`(1M) manual page). In addition, some allocation policies are automatically managed by the `fssweep` and `fsmove` utilities.

The `fssweep` and `fsmove` utilities create allocation policies that have names starting with the characters `fsmove_`.



Each component volume has an allocation policy with a name created by appending the volume name to the string `fsmove_`. For example, a volume named `qoss1a` has an allocation policy named `fsmove_qoss1a`. The file system also has an allocation policy named `fsmove_ALL` that includes all component volumes. VxFS default behavior is to assign `fsmove_ALL` to the file system's mount point as the allocation policy to be used for newly created files.

By convention, `fssweep` and `fsmove` use volume names, so you do not specify the string `fsmove_` when using these utilities.

The `fsmove_ALL` allocation policy allows newly created files to reside on any volume. The order in which volumes are used is by ascending device index numbers. When the `fssweep` utility compares a newly created file's residence against a relocation policy's list of source volumes, `fssweep` treats the files as if they reside on the volume of lowest device index.

When you use the `fssweep` and `fsmove` utilities the first time, the file system can already contain files that have no allocation policies. The `fssweep` utility treats such files as though they reside in the `fsmove_ALL` allocation policy.

If you use the `fssweep` and `fsmove` utilities infrequently, the default allocation policy allows overflow as each volume becomes full. When this happens, it becomes ambiguous where the recently created files really reside.

You can use the `fsapadm` utility to create allocation policies. Avoid using names that would be created by `fsmove`. If a file was moved using `fsapadm` into an allocation policy whose name is something other than the allocation policies used by `fssweep` and `fsmove`, the `fssweep` utility disregards the file when it searches the file system.

You can use the `fsapadm` utility to create a special allocation policy named `fsmove_SITE`. The `fssweep` and `fsmove` utilities never create an allocation policy named `fsmove_SITE`. If this allocation policy exists and was established as the file system's default allocation policy, `fssweep` considers any file in this allocation policy as residing in any of the volumes that belong to the policy. For example, if you create `fsmove_SITE` to specify devices whose index numbers are 4 and 6, and if a relocation policy looks for source volume indexes of 2 and 4, files can be searched. The index 4 is common to both the relocation policy and the allocation policy. The purpose of the `fsmove_SITE` allocation policy is to allow allocating newly created files differently from way the `fsmove_ALL` allocation policy is defined, and still allow the files to be recognizable by the `fssweep` utility.

## Relocation List Format

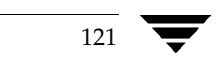
The *relocation list* is standard output from the `fssweep` utility that may be piped to the `fsmove` utility as standard input. The list consists of multiple lines of text, each of which describes one file to be moved and its destination component volume. The following is the format of the text fields:

*safe_filename  destination_volumes*

The *safe_filename* is ordinary text if the file name does not embed special characters or if it is provided on the `fsmove` command line. If special characters are included in the file name that `fssweep` provides to `fsmove`, the exclamation mark (!) escape character delimits the special character. The exclamation marks are followed and preceded by the two hexadecimal digits providing the internal value of the special character. This conversion allows `fsmove` to read special characters from `fssweep` correctly.

The `fssweep` utility provides the *destination_volumes* for the file to be relocated based on the policy manager's configuration policy. If there is more than one destination component volume, `fsmove` tries to relocate a file to each component volume, in the order specified, until the file is successfully written to one of the volumes.





# Quick I/O for Databases 11

VERITAS Quick I/O for Databases (referred to as Quick I/O) lets applications access preallocated VxFS files as raw character devices. This provides the administrative benefits of running databases on file systems without the performance degradation usually associated with databases created on file systems.

Quick I/O is part of the `VRTSvxfs` package, but is available for use only with other VERITAS products. See the *VERITAS Storage Foundation Release Notes* for current product information.

Topics covered in this chapter:





# Quick I/O Functionality and Performance

Many database administrators (DBAs) create databases on file systems because it makes common administrative tasks (such as moving, copying, and backup) much simpler. However, putting databases on file systems significantly reduces database performance. By using VERITAS Quick I/O, you can retain the advantages of having databases on file systems without performance degradation.

Quick I/O uses a special naming convention to allow database applications to access regular files as raw character devices. This provides higher database performance in the following ways:

- Supporting kernel asynchronous I/O
- Supporting direct I/O
- Avoiding kernel write locks
- Avoiding double buffering

## Supporting Kernel Asynchronous I/O

Some operating systems provide kernel support for asynchronous I/O on raw devices, but not on regular files. As a result, even if the database server is capable of using asynchronous I/O, it cannot issue asynchronous I/O requests when the database is built on a file system. Lack of asynchronous I/O significantly degrades performance. Quick I/O lets the database server take advantage of kernel supported asynchronous I/O (through the asyncdsk or Posix AIO interface) on file system files accessed via the Quick I/O interface by providing a character device node that is treated by the OS as a raw device.

## Supporting Direct I/O

I/O on files using `read`() and `write`() system calls typically results in data being copied twice: once between user and kernel space, and later between kernel space and disk. In contrast, I/O on raw devices is direct. That is, data is copied directly between user space and disk, saving one level of copying. As with I/O on raw devices, Quick I/O avoids the extra copying.

### Avoiding Kernel Write Locks

When database I/O is performed via the `write`() system call, each system call acquires and releases a write lock inside the kernel. This lock prevents simultaneous write operations on the same file. Because database systems usually implement their own locks for managing concurrent access to files, write locks unnecessarily serialize I/O operations. Quick I/O bypasses file system locking and lets the database server control data access.

### Avoiding Double Buffering

Most database servers implement their own buffer cache and do not need the system buffer cache. So the memory used by the system buffer cache is wasted, and results in data being cached twice: first in the database cache and then in the system buffer cache. By using direct I/O, Quick I/O does not waste memory on double buffering. This frees up memory that can then be used by the database server buffer cache, leading to increased performance.

## Using VxFS Files as Raw Character Devices

When VxFS with Quick I/O is installed, there are two ways of accessing a file:

- The VxFS interface treats the file as a regular VxFS file
- The Quick I/O interface treats the same file as if it were a raw character device, having performance similar to a raw device

This allows a database server to use the Quick I/O interface while a backup server uses the VxFS interface.

### Quick I/O Naming Convention

To treat a file as a raw character device, Quick I/O requires a file name extension to create an alias for a regular VxFS file. Quick I/O recognizes the alias when you add the following suffix to a file name:

```
::cdev:vxfs:
```

Whenever an application opens an existing VxFS file with the suffix `::cdev:vxfs` (the *cdev* portion is an acronym for *character device*), Quick I/O treats the file as if it were a raw device. For example, if the file *xxx* is a regular VxFS file, then an application can access *xxx* as a raw character device by opening it with the name:

```
xxx::cdev:vxfs:
```



**Note** When Quick I/O is enabled, you cannot create a regular VxFS file with a name that uses the `::cdev:vxfs:` extension. If an application tries to create a regular file named *xxx*`::cdev:vxfs:`, the create fails. If Quick I/O is not available, it is possible to create a regular file with the `::cdev:vxfs:` extension, but this could cause problems if Quick I/O is later enabled. It is advisable to reserve the extension only for Quick I/O files.

## Use Restrictions

There are restrictions to using regular VxFS files as Quick I/O files.

- The name *xxx*`::cdev:vxfs:` is recognized as a special name by VxFS only when:
  - **a.** VxFS with Quick I/O has a valid license
  - **b.** the regular file *xxx* is physically present on the VxFS file system
  - **c.** there is no regular file named *xxx*`::cdev:vxfs:` on the system
- If the file *xxx* is being used for memory mapped I/O, it cannot be accessed as a Quick I/O file.
- An I/O fails if the file *xxx* has a logical hole and the I/O is done to that hole on *xxx*`::cdev:vxfs:`.
- The size of the file cannot be extended by writes through the Quick I/O interface.

# Creating a Quick I/O File Using qiomkfile

The best way to make regular files accessible to the Quick I/O interface and preallocate space for them is to use the `qiomkfile` command. Unlike the VxFS `setext` command, which requires superuser privileges, any user who has read/write permissions can run `qiomkfile` to create the files. The `qiomkfile` command has five options:

| | |
|---|---|
| `-a` | Creates a symbolic link with an absolute path name for a specified file. The default is to create a symbolic link with a relative path name. |
| `-e` | (For Oracle database files to allow tablespace resizing.) Extends the file size by the specified amount. |
| `-h` | (For Oracle database files.) Creates a file with additional space allocated for the Oracle header. |
| `-r` | (For Oracle database files to allow tablespace resizing.) Increases the file to the specified size. |
| `-s` | Preallocates space for a file. |

You can specify file size in terms of bytes (the default), or in kilobytes, megabytes, gigabytes, sectors (512 bytes), or terabytes by adding a `k`, `K`, `m`, `M`, `g`, `G`, `s`, `S`, `t`, or `T` suffix. If the size of the file including the header is not a multiple of the file system block size, it is rounded to a multiple of the file system block size before preallocation.

The `qiomkfile` command creates two files: a regular file with preallocated, contiguous space; and a symbolic link pointing to the Quick I/O name extension. For example, to create a 100 MB file named `dbfile` in `/database`, enter:

```
$ qiomkfile -s 100m /database/dbfile
```

In this example, the first file created is a regular file named `/database/.dbfile` (which has the real space allocated).

The second file is a symbolic link named `/database/dbfile`. This is a relative link to `/database/.dbfile` via the Quick I/O interface, that is, to `.dbfile::cdev:vxfs:`. This allows `.dbfile` to be accessed by any database or application as a raw character device. To check the results, enter:

```
$ ls -al
-rw-r--r--   1 oracle   dba 104857600 Oct 22 15:03 .dbfile
lrwxrwxrwx   1 oracle   dba 19 Oct 22 15:03 dbfile -> \
             .dbfile::cdev:vxfs:
```

or:

```
$ ls -lL
crw-r----- 1oracle   dba 43,0 Oct 22 15:04 dbfile
-rw-r--r-- 1oracle   dba 10485760 Oct 22 15:04 .dbfile
```



If you specify the `-a` option to `qiomkfile`, an absolute path name (see "Using Absolute or Relative Path Names" on page 128) is used so `/database/dbfile` points to `/database/.dbfile::cdev:vxfs:`. To check the results, enter:

```
$ ls -al
-rw-r--r--   1 oracle   dba 104857600 Oct 22 15:05 .dbfile
lrwxrwxrwx   1 oracle   dba 31 Oct 22 15:05 dbfile ->
              /database/.dbfile::cdev:vxfs:
```

See the `qiomkfile`(1) manual page for more information.

# Accessing Regular VxFS Files Through Symbolic Links

Another way to use Quick I/O is to create a symbolic link for each file in your database and use the symbolic link to access the regular files as Quick I/O files.

The following commands create a 100 MB Quick I/O file named `dbfile` on the VxFS file system `/database`. The `dd` command preallocates the file space:

```
$ cd /database
$ dd if=/dev/zero of=/database/.dbfile bs=128k count=800
$ ln -s .dbfile::cdev:vxfs: /database/dbfile
```

Any database or application can then access the file `dbfile` as a raw character device. See the VERITAS Editions product documentation for more information.

## Using Absolute or Relative Path Names

It is usually better to use relative path names instead of absolute path names when creating symbolic links to access regular files as Quick I/O files. Using relative path names prevents copies of the symbolic link from referring to the original file. This is important if you are backing up or moving database files with a command that preserves the symbolic link. However, some applications, such as SAP, require absolute path names.

If you create a symbolic link using a relative path name, both the symbolic link and the file are under the same parent directory. If you want to relocate the file, both the file and the symbolic link must be moved.

It is also possible to use the absolute path name when creating a symbolic link. If the database file is relocated to another directory, you must change the symbolic link to use the new absolute path. You can put all the symbolic links in a directory separate from the data directories. For example, you can create a directory named `/database` and put in all the symbolic links, with the symbolic links pointing to absolute path names.

### Preallocating Files Using the setext Command

You can use the VxFS `setext` command to preallocate file space, but the `setext` command requires superuser privileges. You may need to use the `chown` and `chgrp` commands to change the owner and group permissions on the file after it is created. The following example shows how to use `setext` to create a 100 MB database file for an Oracle database:

```
# cd /database
# touch .dbfile
# setext -r 102400 -f noreserve -f chgsize .dbfile
# ln -s .dbfile::cdev:vxfs: dbfile
# chown oracle dbfile
# chgrp dba dbfile
```

See the `setext`(1M) manual page for more information.

## Using Quick I/O with Oracle Databases

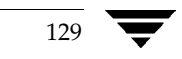
The following example shows how a file can be used by an Oracle database to create a tablespace. This command would be run by the Oracle DBA (typically user ID `oracle`):

```
$ qiomkfile -h headersize -s 100m /database/dbfile
$ sqlplus /nolog
SQL> connect / as sysdba
SQL> create tablespace ts1 datafile '/database/dbfile' size 100M;
SQL> exit;
```

The following example shows how the file can be used by an Oracle database to create a tablespace. Oracle requires additional space for one Oracle header size. So in this example, although 100 MB was allocated to `/database/dbfile`, the Oracle database can use only up to 100 MB minus the Oracle parameter `db_block_size`.

```
$ sqlplus /nolog
SQL> connect / as sysdba
SQL> create tablespace ts1 datafile '/database/dbfile' size 99M;
SQL> exit;
```





# Using Quick I/O with Sybase Databases

Quick I/O works similarly on Sybase database devices.

To create a new database device, preallocate space on the file system by using the `qiomkfile` command, then use the Sybase `buildmaster` command for a master device, or the Transact SQL `disk init` command for a database device. `qiomkfile` creates two files: a regular file using preallocated, contiguous space, and a symbolic link pointing to the `::cdev:vxfs:` name extension. For example, to create a 100 megabyte master device `masterdev` on the file system `/sybmaster`, enter:

```
$ cd /sybmaster
$ qiomkfile -s 100m masterdev
```

You can use this master device while running the `sybsetup` program or `sybinit` script. If you are creating the master device directly, type:

```
$ buildmaster -d masterdev -s 51200
```

To add a new 500 megabyte database device `datadev` to the file system `/sybdata` on your dataserver, enter:

```
$ cd /sybdata
$ qiomkfile -s 500m datadev
...
$ isql -U sa -P sa_password -S dataserver_name
1> disk init
2> name = "logical_name",
3> physname = "/sybdata/datadev",
4> vdevno = "device_number",
5> size = 256000
6> go
```

# Enabling and Disabling Quick I/O

If the Quick I/O feature is licensed and installed, Quick I/O is enabled by default when a file system is mounted. Alternatively, the VxFS `mount -o qio` command enables Quick I/O. The `mount -o noqio` command disables Quick I/O.

If Quick I/O is not installed or licensed, a file system mounts by default without Quick I/O and no error message is displayed. However, if you specify the `-o qio` option, the `mount` command terminates without mounting the file system.

# Cached Quick I/O For Databases

32-bit applications (such as 32-bit databases) can use a maximum of only 4 GB of memory because of the 32-bit address limitation. The Cached Quick I/O feature improves database performance on machines with sufficient memory by also using the file system cache to store data.

For read operations through the Quick I/O interface, data is cached in the system buffer cache, so subsequent reads of the same data can access this cached copy and avoid doing disk I/O. To maintain the correct data in its buffer for write operations, Cached Quick I/O keeps the buffer cache in sync with the data written to disk.

With 64-bit applications, for which limited memory is not a critical problem, using the file system cache still provides performance benefits by using the *read-ahead* functionality. Because of the read-ahead functionality, sequential table scans will benefit the most from using Cached Quick I/O by significantly reducing the query response time.

To use this feature, set the `qio_cache_enable` system parameter with the `vxtunefs` utility, and use the `qioadmin` command to turn the per-file cache advisory on or off. See the `vxtunefs`(1M) and `qioadmin`(1) man pages for more information.



## Enabling Cached Quick I/O

Caching for Quick I/O files can be enabled online when the database is running. You enable caching in two steps:

1. Set the `qio_cache_enable` parameter of `vxtunefs` to enable caching on a file system.

2. Enable the Cached Quick I/O feature for specific files using the `qioadmin` command.

**Note** Quick I/O must be enabled on the file system for Cached Quick I/O to operate.

### Enabling Cached Quick I/O for File Systems

Caching is initially disabled on a file system. You enable Cached Quick I/O for a file system by setting the `qio_cache_enable` option of the `vxtunefs` command after the file system is mounted. For example, to enable Cached Quick I/O for the file system `/database01`, enter:

```
# vxtunefs -s -o qio_cache_enable=1 /database01
```

where `/database01` is a VxFS file system containing the Quick I/O files.

**Note** This command enables caching for all the Quick I/O files on this file system.

You can make this setting persistent across mounts by adding a file system entry in the file `/etc/vx/tunefstab`. For example:

```
/dev/vx/dsk/datadg/database01 qio_cache_enable=1
/dev/vx/dsk/datadg/database02 qio_cache_enable=1
```

For information on how to add tuning parameters, see the `tunefstab`(4) manual page.

## Enabling Cached Quick I/O for Individual Files

There are several ways to enable caching for a Quick I/O file. Use the following syntax to enable caching on an individual file:

```
$ qioadmin -S filename=on mount_point
```

To enable caching for the Quick I/O file `/database01/names.dbf`, type:

```
$ qioadmin -S names.dbf=ON /database01
```

To disable the caching for that file, enter:

```
$ qioadmin -S names.dbf=OFF /database01
```

To make the setting persistent across mounts, create a *qiotab* file, `/etc/vx/qioadmin`, to list files and their caching advisories. Based on the following example, the file `/database/sell.dbf` will have caching turned on whenever the file system `/database` is mounted:

```
device=/dev/vx/dsk/datadg/database01
dates.dbf,off
names.dbf,off
sell.dbf,on
```

**Note** The cache advisories operate only if Cached Quick I/O is enabled for the file system. If the `qio_cache_enable` flag is zero, Cached Quick I/O is `OFF` for all the files in that file system even if the individual file cache advisory for a file is `ON`.

To check on the current cache advisory settings for a file, enter:

```
$ qioadmin -P names.dbf /database01
names.dbf,OFF
```

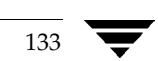
To check the setting of the `qio_cache_enable` flag for a file system, enter:

```
$ vxtunefs -p /database01
qio_cache_enable = 1
```

For more information on the format of the `/etc/vx/qioadmin` file and the command syntax, see the `qioadmin`(1) manual page.

**Note** Check the setting of the flag `qio_cache_enable` using the `vxtunefs` command, and the individual cache advisories for each file, to verify caching.





### Tuning Cached Quick I/O

Not all database files can take advantage of caching. Performance may even degrade in some instances (due to double buffering, for example). Determining which files and applications can benefit from Cached Quick I/O requires that you first collect and analyze the caching statistics.

See the `qiostat`(1) man page for information on gathering statistics, and the VERITAS Editions products documentation for a description of the Cached Quick I/O tuning methodology.

## Quick I/O Statistics

Quick I/O provides the `qiostat` utility to collect database I/O statistics generated over a period of time. `qiostat` reports statistics such as the number of read and write operations, the number of blocks read or written, and the average time spent on read and write operations during an interval. See the `qiostat`(1) manual page for more information.

## Quick I/O Summary

To increase database performance on a VxFS file system using Quick I/O:

- Make sure that the VERITAS Editions product is installed:

  ```
  # swinstall | grep VRTSdbed
  ```

- Make sure that the VERITAS Quick I/O package is licensed:

  ```
  # vxlicrep | grep VXFDD
  ```

- Create a regular VxFS file and preallocate it to required size, or use the `qiomkfile` command. The size of this preallocation depends on the size requirement of the database server.
- Create and access the database using the file name *xxx*`::cdev:vxfs:.`

For information on how to configure VxFS and set up file devices for use with new and existing Oracle databases, see the VERITAS Editions product documentation, and the `qioadmin`(1) and `vxtunefs`(1M) manual pages.

# VERITAS File System Quick Reference A

This appendix lists the VERITAS File System (VxFS) commands and manual pages.

- Command Summary
- Online Manual Pages

This appendix provides instructions and examples on performing the following VxFS operations:

- Creating a File System
- Mounting a File System
- Unmounting a File System
- Displaying Information on Mounted File Systems
- Identifying File System Types
- Resizing a File System
- Backing Up and Restoring a File System
- Using Quotas



# Command Summary

Symbolic links to all VxFS command executables are installed in the `/opt/VRTS/bin` directory. Add this directory to the end of your PATH environment variable to access the commands.

| Command | Description |
|---|---|
| `cfscluster`[3] | CFS cluster configuration command. |
| `cfsdgadm`[3] | Adds or deletes shared disk groups to/from a cluster configuration. |
| `cfsmntadm`[3] | Adds, deletes, modifies, and sets policy on cluster mounted file systems. |
| `cfsmount`, `cfsumount`[3] | Mounts or unmounts a cluster file system. |
| `df` | Reports the number of free disk blocks and inodes for a VxFS file system. |
| `diskusg` | Generates VxFS disk accounting data by user ID. |
| `extentfs` | Extends the size of a VxFS file system. |
| `fcladm` | VxFS File Change Log administration utility. |
| `ff` | Lists file names and inode information for a VxFS file system. |
| `fsadm` | Resizes or defragments a VxFS file system. |
| `fsapadm` | VxFS allocation policy administration utility. |
| `fscat` | Cats a VxFS file system. |
| `fscdsconv` | Converts the byte order of a file system. |
| `fscdstask` | Performs various CDS operations. |
| `fsck` | Checks and repairs a VxFS file system. |
| `fsckpt_restore`[5] | VxFS Storage Checkpoint file system restoration utility. |
| `fsckptadm` | VxFS Storage Checkpoint administration utility. |
| `fsclustadm`[3] | Manages cluster-mounted VxFS file systems. |
| `fsdb` | VxFS file system debugger. |

| Command | Description |
|---|---|
| fsdbencap | Database encapsulation utility. |
| fsmove[4, 5] | Relocates files to a destination component volume. |
| fsrpadm | VxFS relocation policy administration utility. |
| fssweep[4, 5] | Sweeps a multiple-volume VxFS file system for files to relocate. |
| fstyp | Returns the type of file system on a specified disk partition. |
| fsvoladm | VxFS device administration utility. |
| getext | Gets extent attributes for a VxFS file system. |
| glmconfig[3] | Group Lock Manager (GLM) configuration utility. |
| mkfs | Constructs a VxFS file system. |
| mount | Mounts a VxFS file system. |
| ncheck | Generates path names from inode numbers for a VxFS file system. |
| newfs | Creates a new VxFS file system. |
| qioadmin[1] | VxFS Quick I/O for Databases cache administration utility. |
| qiomkfile[1] | Creates a VxFS Quick I/O device file. |
| qiostat[1] | VxFS Quick I/O for Databases statistics utility. |
| quot | Summarizes ownership on a VxFS file system. |
| quotacheck | Checks VxFS file system quota consistency. |
| rvxdump | Incremental file system dump. |
| rvxrestore | Restores a file system incrementally. |
| setext | Sets extent attributes on a file in a VxFS file system. |
| vxdump | Incremental file system dump. |
| vxenablef | Enables specific VxFS features. |
| vxfsconvert | Converts an unmounted file system to VxFS or upgrades a VxFS disk layout version. |



| Command | Description |
|---|---|
| `vxfsstat` | Displays file system statistics. |
| `vxlsino` | VxFS reverse path name lookup utility. |
| `vxrestore` | Restores a file system incrementally. |
| `vxtunefs` | Tunes a VxFS file system. |
| `vxumount` | Unmounts a VxFS file system. |
| `vxupgrade` | Upgrades the disk layout of a mounted VxFS file system. |

[1] Functionality available only with VERITAS Quick I/O for Databases feature
[3] Functionality available only with the VERITAS Cluster File System product
[4] Functionality available only with the VERITAS Quality of Storage Service option
[5] New in VxFS 4.1

# Online Manual Pages

This release includes the following online manual pages as part of the `VRTSvxfs` package. These are installed in the appropriate directories under `/opt/VRTS/man` (add this to your MANPATH environment variable), but does not update the windex database. To ensure that new VxFS manual pages display correctly, update the windex database after installing `VRTSvxfs`. See the `catman`(1M) manual page for more information.

| **Section 1** | **Description** |
|---|---|
| qioadmin[1] | VxFS Quick I/O for Databases cache administration utility. |
| qiomkfile[1] | Creates a VxFS Quick I/O device file. |
| qiostat[1] | VxFS Quick I/O for Databases statistics utility. |
| **Section 1M** | **Description** |
| cfscluster[3] | CFS cluster configuration command. |
| cfsdgadm[3] | Adds or deletes shared disk groups to/from a cluster configuration. |
| cfsmntadm[3] | Adds, deletes, modifies, and sets policy on cluster mounted file systems. |
| cfsmount, cfsumount[3] | Mounts or unmounts a cluster file system. |
| `df_vxfs` | Reports the number of free disk blocks and inodes for a VxFS file system. |
| extendfs_vxfs | Extends the size of a VxFS file system. |
| `fcladm` | VxFS File Change Log administration utility. |
| ff_vxfs | Lists file names and inode information for a VxFS file system. |
| `fsrpadm` | VxFS relocation policy administration utility. |
| `fsadm_vxfs` | Resizes or reorganizes a VxFS file system. |
| `fsapadm` | VxFS allocation policy administration utility. |
| `fscat_vxfs` | Cats a VxFS file system. |
| `fscdsconv` | Converts the byte order of a file system. |
| `fscdstask` | Performs various CDS operations. |



| | |
|---|---|
| `fsck_vxfs` | Checks and repairs a VxFS file system. |
| fsckptadm | VxFS Storage Checkpoint administration utility. |
| fsckpt_restore[5] | VxFS Storage Checkpoint restore utility. |
| fsclustadm[3] | Manages cluster-mounted VxFS file systems. |
| `fsvoladm` | VxFS device administration utility. |
| `fsdb_vxfs` | VxFS file system debugger. |
| fsmove[4, 5] | Relocates files to a destination component volume. |
| `fsrpadm` | VxFS relocation policy administration command. |
| fssweep[4, 5] | Traverses the directory structure of one or more file systems. |
| `fsvoladm` | VxFS volume administration utility. |
| `getext` | Gets extent attributes for a VxFS file system. |
| `glmconfig`[3] | Group Lock Manager (GLM) configuration utility. |
| `mkfs_vxfs` | Constructs a VxFS file system. |
| `mount_vxfs` | Mounts a VxFS file system. |
| `ncheck_vxfs` | Generates path names from inode numbers for a VxFS file system. |
| `newfs_vxfs` | Creates a new VxFS file system. |
| `quot` | Summarizes ownership on a VxFS file system. |
| `quotacheck_vxfs` | Checks VxFS file system quota consistency. |
| setext | Sets extent attributes on a file in a VxFS file system. |
| `vxdiskusg` | Generates VxFS disk accounting data by user ID. |
| `vxdump` | Incremental file system dump. |
| vxenablef | Enables specific VxFS features. |
| `vxfsconvert` | Converts an unmounted file system to VxFS or upgrades a VxFS disk layout version. |
| `vxfsstat` | Displays file system statistics. |

| | |
|---|---|
| vxlsino[5] | VxFS reverse path name lookup utility. |
| vxrestore | Restores a file system incrementally. |
| vxtunefs | Tunes a VxFS file system. |
| vxupgrade | Upgrades the disk layout of a mounted VxFS file system. |
| vxumount | Unmounts a VxFS file system. |
| **Section 3** | **Description** |
| fsckpt_clearcontext | Clears current process Storage Checkpoint context. |
| fsckpt_cntl | Performs control functions and Storage Checkpoint state changes. |
| fsckpt_create | Creates a Storage Checkpoint associated with a file system handle. |
| fsckpt_createall | Creates a Storage Checkpoint associated with a list of file system handles. |
| fsckpt_fbmap[5] | Retrieves the block map from a Storage Checkpoint file. |
| fsckpt_fclose | Closes a Storage Checkpoint file. |
| fsckpt_finfo | Returns the status information from a Storage Checkpoint file. |
| fsckpt_fopen | Opens a Storage Checkpoint file. |
| fsckpt_fpromote[5] | Promote a file from a Storage Checkpoint into another fileset. |
| fsckpt_fsclose | Closes a mount point opened for Storage Checkpoint management. |
| fsckpt_fsopen | Opens a mount point for Storage Checkpoint management. |
| fsckpt_info | Returns status information on a Storage Checkpoint. |
| fsckpt_intro | Introduction to the VxFS file system Checkpointing API. |
| fsckpt_mkprimary[5] | Make a Storage Checkpoint in a VxFS file system the primary fileset for that file system. |
| fsckpt_opts_create[5] | Creates a Storage Checkpoint associated with a file system handle. |
| fsckpt_opts_createall[5] | Creates a Storage Checkpoint associated with a list of file system handles. |
| fsckpt_remove | Removes a Storage Checkpoint from a file system handle. |



| | |
|---|---|
| fsckpt_rename | Renames a Storage Checkpoint from a file system handle. |
| fsckpt_sa_info[5] | Return status information on a Storage Checkpoint within a VxFS file system on a block special device. |
| fsckpt_setcontext[5] | Turns Storage Checkpoint quotas on or off. |
| fsckpt_setqlimit[5]<br>fsckpt_qetqlimit[5] | Sets or gets Storage Checkpoint quota limits. |
| fsckpt_setquota[5] | Turns Storage Checkpoint quotas on or off. |
| `vxfs_ap_assign_ckpt` | Assigns an allocation policy to file data in a Storage Checkpoint. |
| `vxfs_ap_assign_file` | Assigns an allocation policy for file data and metadata. |
| `vxfs_ap_assign_fs` | Assigns an allocation policy for all file data and metadata within a specified file system. |
| `vxfs_ap_define` | Defines a new allocation policy. |
| `vxfs_ap_enforce_file` | Ensures that all blocks in a specified file match the file allocation policy. |
| `vxfs_ap_enumerate` | Returns information about all allocation policies. |
| `vxfs_ap_query` | Returns information about a specific allocation policy. |
| `vxfs_ap_query_ckpt` | Returns information about allocation policies for each Storage Checkpoint. |
| `vxfs_ap_query_file` | Returns information about allocation policies assigned to a specified file. |
| `vxfs_ap_query_fs` | Retrieves allocation policies assigned to a specified file system. |
| `vxfs_ap_remove` | Deletes a specified allocation policy. |
| `vxfs_fcl_sync` | Sets a synchronization point in the VxFS File Change Log. |
| `vxfs_get_ioffsets` | Obtains VxFS inode field offsets. |
| `vxfs_inotopath` | Returns path names for a given inode number. |
| `vxfs_nattr_link` | Links to a named data stream. |

| | |
|---|---|
| vxfs_nattr_open | Opens a named data stream. |
| vxfs_nattr_rename | Renames a named data stream. |
| vxfs_nattr_unlink | Removes a named data stream. |
| vxfs_nattr_utimes | Sets access and modification times for named data streams. |
| vxfs_vol_add | Adds a volume to a multi-volume file system. |
| vxfs_vol_deencapsulate | De-encapsulates a volume from a multi-volume file system. |
| vxfs_vol_encapsulate | Encapsulates a volume within a multi-volume file system. |
| vxfs_vol_encapsulate_bias | Encapsulates a volume within a multi-volume file system. |
| vxfs_vol_enumerate | Returns information about the volumes within a multi-volume file system. |
| vxfs_vol_remove | Removes a volume from a multi-volume file system. |
| vxfs_vol_resize | Resizes a specific volume within a multi-volume file system. |
| vxfs_vol_stat | Returns free space information about a component volume within a multi-volume file system. |
| **Section 4** | **Description** |
| fs_vxfs | Format of a VxFS file system volume. |
| inode_vxfs | Format of a VxFS file system inode. |
| tunefstab | VxFS file system tuning parameters table. |
| **Section 7** | **Description** |
| vxfsio | VxFS file system control functions. |

[1] Functionality available only with VERITAS Quick I/O for Databases feature
[3] Functionality available only with the VERITAS Cluster File System product
[4] Functionality available only with the VERITAS Quality of Storage Service option
[5] New in VxFS 4.1



# Creating a File System

The `mkfs` command creates a VxFS file system by writing to a special character device file. The special character device is a raw disk device or a VERITAS Volume Manager (VxVM) volume. `mkfs` builds a file system with a root directory and a `lost+found` directory.

Before running `mkfs`, you must create the target device. Refer to your operating system documentation for more information. If you are using a logical device (such as a VxVM volume), see the VxVM documentation for instructions on device initialization.

## How to Create a File System

To create a file system, use the `mkfs` command:

```
mkfs [-F vxfs] [generic_options] [-o specific_options] special [size]
```

| | |
|---|---|
| `vxfs` | The file system type. |
| *generic_options* | Options common to most other file system types. |
| *specific_options* | Options specific to VxFS. |
| `-o N` | Displays the geometry of the file system and does not write to the device. |
| `-o largefiles` | Allows users to create files larger than two gigabytes. The default option is `largefiles`. |
| *special* | The character (raw) device or VERITAS Volume Manager volume. |
| *size* | The size of the new file system (in sectors). |

See the following manual pages for more information about creating VxFS file systems:

- `mkfs`(1M)
- `mkfs_vxfs`(1M)

**Example**

To create a VxFS file system 12288 sectors in size on `VxVM volume`, enter:

```
# mkfs -F vxfs /dev/vx/rdsk/diskgroup/volume 12288
```

Information similar to the following displays:

```
version 6 layout
```

```
262144 sectors, 262144 blocks of size 1024, log size 1024 blocks
largefiles supported
```

At this point, you can mount the newly created file system.

# Converting a UFS File System to VxFS

The `vxfsconvert` command can be used to convert a UFS file system to a VxFS file system.

## How to Convert a File System

To convert a UFS file system, use the `vxfsconvert` command:

```
vxfsconvert [-l logsize] [-s size] [-efnNvyY] special
```

| | |
|---|---|
| -e | Estimates the amount of space required to complete the conversion. |
| -f | Displays the list of supported file system types. |
| `-l` *logsize* | Specifies the size of the file system intent log. |
| -n\|N | Assumes a `no` response to all questions asked by `vxfsconvert`. |
| `-s` *size* | Directs `vxfsconvert` to use free disk space past the current end of the file system to store VxFS metadata. |
| -v | Specifies verbose mode. |
| -y\|Y | Assumes a `yes` response to all questions asked by `vxfsconvert`. |

*special*

See the `vxfsconvert`(1M) manual page for more information about converting a UFS file system to a VxFS file system.

**Example**

To convert a UFS file system to a VxFS file system with an intent log size of 4096 blocks, enter:

```
# vxfsconvert -l 4096 /dev/vx/rdsk/diskgroup/volume
```



# Mounting a File System

You can mount a VxFS file system by using the `mount` command. If you enter this command, the generic `mount` command parses the arguments and the `-F` *fstype* option executes the `mount` command specific to that file system type. For VxFS and VERITAS-installed products, the generic `mount` command executes the VxFS mount command from the directory `/sbin/fs/vxfs3.5`. If the `-F` option is not supplied, the command searches the file `/etc/fstab` for a file system and an *fstype* matching the special file or mount point provided. If no file system type is specified, `mount` uses the default file system type (VxFS).

## How to Mount a File System

After you create a VxFS file system, you can use the `mount` command to mount the file system:

```
mount [-F vxfs] [generic_options] [-r] [-o specific_options] \
special mount_point
```

| | |
|---|---|
| `vxfs` | File system type. |
| *generic_options* | Options common to most other file system types. |
| *specific_options* | Options specific to VxFS. |
| `-o ckpt=`*ckpt_name* | Mounts a VERITAS Storage Checkpoint. |
| `-o cluster` | Mounts a file system in shared mode. Available only with the VxFS cluster file system feature. |
| *special* | Block special device. |
| *mount_point* | Directory on which to mount the file system. |
| `-r` | Mounts the file system as read-only. |

## Mount Options

The `mount` command has numerous options to tailor a file system for various functions and environments. Some *specific_options* are listed below.

- Security feature

  If security is important, use `blkclear` to ensure that deleted files are completely erased before the space is reused.

- Support for large files

  If you specify the `largefiles` option, you can create files larger than two gigabytes on the file system. The default option is `largefiles`.

- Support for cluster file systems

  If you specify the `cluster` option, the file system is mounted in shared mode. Cluster file systems depend on several other VERITAS products that must be correctly configured before a complete clustering environment is enabled.

- Using Storage Checkpoints

  The `ckpt=`*checkpoint_name* option mounts a Storage Checkpoint of a mounted file system that was previously created by the `fsckptadm` command.

- Using Storage Checkpoints

  The `ckpt=`*checkpoint_name* option mounts a Storage Checkpoint of a mounted file system that was previously created by the `fsckptadm` command.

- Using databases

  If you are using databases with VxFS and if you have installed a license key for the VERITAS Quick I/O for Databases feature, the `mount` command enables Quick I/O by default (the same as specifying the `qio` option). The `noqio` option disables Quick I/O. If you do not have Quick I/O, `mount` ignores the `qio` option. Alternatively, you can increase database performance using the mount option `convosync=direct`, which utilizes direct I/O. See "Quick I/O for Databases" on page 123 for more information.

- News file systems

  If you are using `cnews`, use `delaylog` (or `tmplog`),`mincache=closesync` because `cnews` does an `fsync()` on each news file before marking it received. The `fsync()` is performed synchronously as required, but other options are delayed.

- Temporary file systems

  For a temporary file system such as `/tmp`, where performance is more important than data integrity, use `tmplog,mincache=tmpcache`.



See "Choosing mount Command Options" on page 23 and the following manual pages for more information about the `mount` command and its available options:

`fstab`(4)

`fsckptadm`(1M)

`mount`(1M)

`mount_vxfs`(1M)

**Example**

To mount the file system **`/dev/vx/dsk/fsvol/vol1`** on the `/ext` directory with read/write access and delayed logging, enter:

```
# mount -F vxfs -o delaylog /dev/vx/dsk/fsvol/vol1 /ext
```

## How to Edit the fstab File

You can edit the `/etc/fstab` file to automatically mount a file system at boot time. You must specify:

- the special block device name to `mount`
- the mount point
- the file system type (`vxfs`)
- the `mount` options
- the backup frequency
- which `fsck` pass looks at the file system

Each entry must be on a single line. See the `fstab`(4) manual page for more information about the `/etc/fstab` file format.

Here is a typical `fstab` file with the new file system on the last line:

```
# System /etc/fstab file.  Static
# information about the file systems
# See fstab(4) and sam(1M) for further
# details on configuring devices.
/dev/vg00/lvol3    / vxfs delaylog 0 1
/dev/vg00/lvol1    /stand hfs defaults 0 1
/dev/vg00/lvol4    /tmp vxfs delaylog 0 2
/dev/vg00/lvol5    /home vxfs delaylog 0 2
/dev/vg00/lvol6    /opt vxfs delaylog 0 2
/dev/vg00/lvol7    /usr vxfs delaylog 0 2
/dev/vg00/lvol8    /var vxfs delaylog 0 2
/dev/vx/dsk/fsvol  /ext vxfs delaylog 0 2
```



# Unmounting a File System

Use the `umount` command to unmount a currently mounted file system.

## How to Unmount a File System

To unmount a file system, use the following syntax:

```
umount special | mount_point
```

Specify the file system to be unmounted as a *mount_point* or *special* (the device on which the file system resides). See the `umount_vxfs`(1M) manual page for more information about this command and its available options.

**Example**

To unmount the file system `/dev/vx/dsk/fsvol/vol1`, enter:

```
# umount /dev/vx/dsk/fsvol/vol1
```

To unmount all file systems not required by the system, enter:

```
# umount -a
```

This unmounts all file systems except `/` (root), `/usr`, `/var`, `/opt`, and `/tmp`.

# Displaying Information on Mounted File Systems

You can use the `mount` command to display a list of currently mounted file systems.

## How to Display File System Information

To view the status of mounted file systems, use the syntax:

```
mount -v
```

This shows the file system type and `mount` options for all mounted file systems. The `-v` option specifies verbose mode.

See the following manual pages for more information about the `mount` command and its available options:

`mount`(1M)

`mount_vxfs`(1M)

**Example**

When invoked without options, the `mount` command displays file system information similar to the following:

```
# mount
  /dev/vg00/lvol3 on / type vxfs ioerror=mwdisable,delaylog Wed Jun 5 \
  3:23:40 2004
/dev/vg00/lvol8 on /var type vxfs ioerror=mwdisable,delaylog Wed Jun 5 \
  3:23:56 2004
/dev/vg00/lvol7 on /usr type vxfs ioerror=mwdisable,delaylog Wed Jun 5 \
  3:23:56 2004
/dev/vg00/lvol6 on /tmp type vxfs ioerror=mwdisable,delaylog Wed Jun 5 \
  3:23:56 2004
/dev/vg00/lvol5 on /opt type vxfs ioerror=mwdisable,delaylog Wed Jun 5 \
  3:23:57 2004
/dev/vg00/lvol1 on /stand type hfs defaults on Thu Jun 6 4:17:20 2004 \
/dev/vgdb/lvol13 on /oracle type vxfs ioerror=mwdisable,delaylog Thu \
  Jun 6 4:17:20 2004
/dev/vg00/lvol4 on /home type vxfs ioerror=mwdisable,delaylog on Thu \
  Jun 6  4:17:20 2004
/dev/vgdb/lvol9 on /bench type vxfs ioerror=mwdisable,delaylog on Thu  \
  Jun 6 4:17:11 2004
```



# Identifying File System Types

Use the `fstyp` command to determine the file system type for a specified file system. This is useful when a file system was created elsewhere and you want to know its type.

## How to Identify a File System

To determine the status of mounted file systems, use the syntax:

```
fstyp -v special
```

| | |
|---|---|
| *special* | The character (raw) device. |
| `-v` | Specifies verbose mode. |

See the following manual pages for more information about the `fstyp` command and its available options:

`fstyp`(1M)

**Example**

To find out what kind of file system is on the device `/dev/vx/dsk/fsvol/vol1`, enter:

```
# fstyp -v /dev/vx/dsk/fsvol/vol1
```

The output indicates that the file system type is `vxfs`, and displays file system information similar to the following:

```
vxfs
version: 6
f_bsize: 8192
f_frsize: 1024
f_blocks: 1027432
f_bfree: 1026075
f_bavail: 961946
f_files: 256548
f_ffree: 256516
f_favail: 256516
f_fsid: 520114176
```

```
f_basetype: vxfs
f_namemax: 254
f_magic: a501fcf5
f_featurebits: 0
f_flag: 0
f_fsindex: 7
f_size: 4194304
```

# Resizing a File System

You can extend or shrink mounted VxFS file systems using the `fsadm` command. Use the `extendfs` command to extend the size of an unmounted file system. A file system using the Version 4 disk layout can be up to two terabytes in size. A file system using the Version 5 disk layout can be up to 32 terabytes in size. A file system using the Version 6 disk layout can be up to 256 terabytes in size. The size to which a Version 5 or Version 6 disk layout file system can be increased depends on the file system block size (as shown in the tables under "VxFS Version 5 Disk Layout" on page 210 and "VxFS Version 6 Disk Layout" on page 211). See the following manual pages for more information about resizing file systems:

`extendfs`(1M)

`fsadm_vxfs`(1M)

## How to Extend a File System Using fsadm

If a VxFS file system is not large enough, you can increase its size. The size of the file system is specified in units of 1024-byte blocks (or sectors).

**Note** If a file system is full, busy, or too fragmented, the resize operation may fail.

To extend a VxFS file system, use the syntax:

`fsadm` [-F vxfs] [`-b` *newsize*] [`-r` *rawdev*] *mount_point*

| | |
|---|---|
| `vxfs` | The file system type. |
| *newsize* | The size (in sectors) to which the file system will increase. |
| *mount_point* | The file system's mount point. |
| `-r` *rawdev* | Specifies the path name of the raw device if there is no entry in `/etc/fstab` and `fsadm` cannot determine the raw device. |



**Note** The device must have enough space to contain the larger file system. See the `format`(1M) manual page or the *VERITAS Volume Manager Administrator's Guide* for more information.

**Example**

To extend the VxFS file system mounted on `/ext` to 22528 sectors, enter:

```
# fsadm -F vxfs -b 22528 /ext
```

## How to Shrink a File System

You can decrease the size of the file system using `fsadm`, even while the file system is mounted.

**Note** In cases where data is allocated towards the end of the file system, shrinking may not be possible. If a file system is full, busy, or too fragmented, the resize operation may fail.

To decrease the size of a VxFS file system, use the syntax:

```
fsadm [-F vxfs] [-b newsize] [-r rawdev] mount_point
```

| | |
|---|---|
| `vxfs` | The file system type. |
| *newsize* | The size (in sectors) to which the file system will shrink. |
| *mount_point* | The file system's mount point. |
| `-r` *rawdev* | Specifies the path name of the raw device if there is no entry in `/etc/fstab` and `fsadm` cannot determine the raw device. |

**Example**

To shrink a VxFS file system mounted at `/ext` to 20480 sectors, enter:

```
# fsadm -F vxfs -b 20480 /ext
```

**Caution** After this operation, there is unused space at the end of the device. You can then resize the device, but be careful not to make the device smaller than the new size of the file system.



## How to Reorganize a File System

You can reorganize (or compact) a fragmented file system using `fsadm`, even while the file system is mounted. This may help shrink a file system that could not previously be decreased.

**Note** If a file system is full or busy, the reorg operation may fail.

To reorganize a VxFS file system, use the syntax:

```
fsadm [-F vxfs] [-e] [-d] [-E] [-D] [-r rawdev] mount_point
```

| | |
|---|---|
| `vxfs` | The file system type |
| `-d` | Reorders directory entries to put subdirectory entries first, then all other entries in decreasing order of time of last access. Also compacts directories to remove free space. |
| `-D` | Reports on directory fragmentation. |
| `-e` | Minimizes file system fragmentation. Files are reorganized to have the minimum number of extents. |
| `-E` | Reports on extent fragmentation. |
| *mount_point* | The file system's mount point. |
| `-r` *rawdev* | Specifies the path name of the raw device if there is no entry in `/etc/fstab` and `fsadm` cannot determine the raw device. |

**Example**

To reorganize the VxFS file system mounted at `/ext`, enter:

```
# fsadm -F vxfs -EeDd /ext
```

## How to Extend a File System Using extendfs

If a VxFS file system is not mounted, you can use the `extendfs` command to increase the size of the file system.

To extend a VxFS file system, use the syntax:

```
extendfs [-F vxfs] [-q] [-v] [-s size] special
```

| | |
|---|---|
| `vxfs` | The file system type |
| `-q` | Displays the size of *special* without resizing it |
| `-v` | Specifies verbose mode |
| `-s` *size* | Specifies the number of blocks to add to the file system (maximum if not specified) |
| *special* | Either a logical volume or a disk partition |

**Note** The device must have enough space to hold the new larger file system.

When the file system size is grown with the `extendfs` command, the intent log size is not automatically increased. This issue is most visible when upgrading file systems with disk layout Versions prior to 3 and of a size smaller than 8 MB. When such a file system is upgraded to disk layout Version 4 and then extended to a size greater than 8 MB with the `extendfs` command, the file system cannot be mounted since the minimum allowed intent log size is 256K.

**Example**

To increase the capacity of a file system on a VM volume, enter:

```
# umount /dev/vg00/lvol7
# lvextend -L larger_size /dev/vg00/lvol7
# extendfs -F vxfs /dev/vg00/rlvol7
# mount -F vxfs /dev/vg00/lvol7 mount_point
```



# Backing Up and Restoring a File System

To back up a VxFS file system, you first create a read-only snapshot file system, then back up the snapshot. This procedure lets you keep the main file system on line. The snapshot is a copy of the *snapped* file system that is frozen at the moment the snapshot is created.

See "Online Backup Using File System Snapshots" on page 89 and the following manual pages for more information about the `mount`, `vxdump`, and `vxrestore` commands and their available options:

- `mount`(1M)
- `mount_vxfs`(1M)
- `vxdump`(1M)
- `vxrestore`(1M)

## How to Create and Mount a Snapshot File System

The first step in backing up a VxFS file system is to create and mount a snapshot file system. To create and mount a snapshot of a VxFS file system, use the syntax:

```
mount [-F vxfs] -o snapof=source,[snapsize=size] \
destination snap_mount_point
```

| | |
|---|---|
| *source* | The special device name or mount point of the file system to copy. |
| *destination* | The name of the special device on which to create the snapshot. |
| *size* | The size of the snapshot file system in sectors. |
| *snap_mount_point* | Location where to mount the snapshot; *snap_mount_point* must exist before you enter this command. |

**Example**

To create a snapshot file system of the file system at `/home` on `/dev/vx/dsk/fsvol/vol1` and mount it at `/snapmount`, enter:

```
# mount -F vxfs -o snapof=/home, \
  snapsize=32768 /dev/vx/dsk/fsvol/vol1 /snapmount
```

You can now back up the file system, as described in the following section.

## How to Back Up a File System

After creating a snapshot file system as described in the previous section, you can use `vxdump` to back it up. To back up a VxFS snapshot file system, use the syntax:

```
vxdump [-c] [-f backupdev] snap_mount_point
```

| | |
|---|---|
| `-c` | Specifies using a cartridge tape device. |
| *backupdev* | The device on which to back up the file system. |
| *snap_mount_point* | The snapshot file system's mount point. |

**Example**

To back up the VxFS snapshot file system mounted at `/snapmount` to the tape drive with device name `/dev/rmt`, enter:

```
# vxdump -cf /dev/rmt /snapmount
```

## How to Restore a File System

After backing up the file system, you can restore it using the `vxrestore` command. First, create and mount an empty file system. To restore a VxFS snapshot file system, use the syntax:

```
vxrestore [-v] [-x] [filename]
```

| | |
|---|---|
| `-v` | Specifies verbose mode. |
| `-x` | Extracts the named files from the tape. |
| *filename* | The file or directory to restore. If *filename* is omitted, the root directory (and thus the entire tape) is extracted. |

**Example**

To restore a VxFS snapshot file system from the tape `/dev/st1` into the mount point `/restore`, enter:

```
# cd /restore
# vxrestore -v -x -f /dev/st1
```



# Using Quotas

You can use quotas to allocate per-user quotas on VxFS file systems.

See "Quotas" on page 97 and the following manual pages for more information about the `quota`, `quotaon`, `quotaoff`, and `edquota` commands and their available options:

- `edquota`(1M)
- `quota`(1M)
- `quotaon`(1M)
- `quotaoff`(1M)

## How to Turn On Quotas

You can enable quotas at mount time or after a file system is mounted. The root directory of the file system must contain a file named `quotas` that is owned by `root`.

To turn on quotas for a mounted file system, use the syntax:

```
quotaon mount_point
```

To mount a file system and turn on quotas at the same time, use the syntax:

```
mount -F vxfs -o quota special mount_point
```

If the root directory does not contain a `quotas` file, the `mount` command succeeds, but `quotas` are not turned on.

**Example**

To create a quotas file (if it does not already exist) and turn on quotas for a VxFS file system mounted at `/mnt`, enter:

```
# touch /mnt/quotas
# quotaon /mnt
```

To turn on quotas for a file system at mount time, enter:

```
# mount -F vxfs -o quota /dev/vx/dsk/fsvol/vol1 /mnt
```

## How to Set Up User Quotas

You can set user quotas with the `edquota` command if you have superuser privileges. User quotas can have a *soft limit* and/or *hard limit*. You can modify the limits or assign them specific values. Users are allowed to exceed the soft limit, but only for a specified time. Disk usage can never exceed the hard limit. The default time limit for exceeding the soft limit is seven days on VxFS file systems.

`edquota` creates a temporary file for a specified user. This file contains on-disk quotas for each mounted VxFS file system that has a `quotas` file. The temporary file has one or more lines similar to:

```
fs /mnt blocks (soft = 0, hard = 0) inodes (soft=0, hard=0)
fs /mnt1 blocks (soft = 100, hard = 200) inodes (soft=10, hard=20)
```

Quotas do not need to be turned on for `edquota` to work. However, the quota limits apply only after quotas are turned on for a given file system.

`edquota` has an option to modify time limits. Modified time limits apply to the entire file system; you cannot set time limits for an individual user.

To invoke the quota editor, use the syntax:

```
edquota username
```

To modify the time limit, use the syntax:

```
edquota -t
```



## How to View Quotas

The superuser or individual user can view disk quotas and usage on VxFS file systems using the `quota` command. To view quotas for a specific user, use the syntax:

```
quota -v username
```

This command displays the user's quotas and disk usage on all mounted VxFS file systems where the `quotas` file exists. You will see all established quotas regardless of whether or not the quotas are actually turned on.

## How to Turn Off Quotas

You can turn off quotas for a mounted file system using the `quotaoff` command. To turn off quotas for a file system, use the syntax:

```
quotaoff mount_point
```

**Example**

To turn off quotas for a VxFS file system mounted at `/mnt`, enter:

```
# quotaoff /mnt
```

# Kernel Messages B

This appendix contains a listing of diagnostic or error messages generated by the VERITAS File System (VxFS) kernel. Each message has a description and a suggestion on how to handle or correct the underlying problem.

The following topics are covered in this chapter:





# File System Response to Problems

When the file system encounters problems, it responds in one of three ways:

- Marks an inode bad
- Disables transactions
- Disables the file system

## Marking an Inode Bad

Inodes can be marked bad if an inode update or a directory-block update fails. In these types of failures, the file system does not know what information is on the disk, and considers all the information that it finds to be invalid. After an inode is marked bad, the kernel still permits access to the file name, but any attempt to access the data in the file or change the inode fails.

## Disabling Transactions

If the file system detects an error while writing the intent log, it disables transactions. After transactions are disabled, the files in the file system can still be read or written, but no block or inode frees or allocations, structural changes, directory entry changes, or other changes to metadata are allowed.

## Disabling a File System

If an error occurs that compromises the integrity of the file system, VxFS disables itself. If the intent log fails or an inode-list error occurs, the super-block is ordinarily updated (setting the `VX_FULLFSCK` flag) so that the next `fsck` does a full structural check. If this super-block update fails, any further changes to the file system can cause inconsistencies that are undetectable by the intent log replay. To avoid this situation, the file system disables itself.

## Recovering a Disabled File System

When the file system is disabled, no data can be written to the disk. Although some minor file system operations still work, most simply return `EIO`. The only thing that can be done when the file system is disabled is to do a `umount` and run a full `fsck`.

Although a log replay may produce a clean file system, do a full structural check to be safe. To execute a full structural check, enter:

```
# fsck -F vxfs -o full -y /dev/vx/rdsk/diskgroup/volume
```

**Caution** Be careful when running this command. By specifying the `-y` option, all `fsck` user prompts are answered with a "yes", which can make irreversible changes if it performs a full file system check.

The file system usually becomes disabled because of disk errors. Disk failures that disable a file system should be fixed as quickly as possible (see the `fsck_vxfs`(1M) manual page).

# Kernel Messages

This section lists the VxFS kernel error messages in numerical order. The Description subsection for each message describes the problem, the Action subsection suggests possible solutions.

## Global Message IDs

When a VxFS kernel message displays on the system console, it is preceded by a numerical ID shown in the `msgcnt` field. This ID number increases with each instance of the message to guarantee that the sequence of events is known when analyzing file system problems.

Each message is also written to an internal kernel buffer that you can view in the file `/var/adm/syslog/syslog.log`.

In some cases, additional data is written to the kernel buffer. For example, if an inode is marked bad, the contents of the bad inode are written. When an error message is displayed on the console, you can use the unique message ID to find the message in `/var/adm/syslog/syslog.log` and obtain the additional information.



| Message Number | Message and Definition |
|---|---|
| 001 | NOTICE: msgcnt x: mesg 001: V-2-01: vx_nospace - *mount_point* file system full (*n* block extent)<br>◆ Description<br>The file system is out of space.<br>Often, there is plenty of space and one runaway process used up all the remaining free space. In other cases, the available free space becomes fragmented and unusable for some files.<br>◆ Action<br>Monitor the free space in the file system and prevent it from becoming full. If a runaway process has used up all the space, stop that process, find the files created by the process, and remove them. If the file system is out of space, remove files, defragment, or expand the file system.<br>To remove files, use the `find` command to locate the files that are to be removed. To get the most space with the least amount of work, remove large files or file trees that are no longer needed. To defragment or expand the file system, use `fsadm` (see the `fsadm_vxfs`(1M) manual page). |
| 002 | WARNING: msgcnt x: mesg 002: V-2-02: vx_snap_strategy - *mount_point* file system write attempt to read-only file system<br>WARNING: msgcnt x: mesg 002: V-2-02: vx_snap_copyblk - *mount_point* file system write attempt to read-only file system<br>◆ Description<br>The kernel tried to write to a read-only file system. This is an unlikely problem, but if it occurs, the file system is disabled.<br>◆ Action<br>The file system was not written, so no action is required. Report this as a bug to your customer support organization. |

| Message Number | Message and Definition |
|---|---|
| 003, 004, 005 | WARNING: msgcnt x: mesg 003: V-2-03: vx_mapbad - *mount_point* file system free extent bitmap in au aun marked bad<br>WARNING: msgcnt x: mesg 004: V-2-04: vx_mapbad - *mount_point* file system free inode bitmap in au aun marked bad<br>WARNING: msgcnt x: mesg 005: V-2-05: vx_mapbad - *mount_point* file system inode extended operation bitmap in au aun marked bad<br>◆ Description<br>If there is an I/O failure while writing a bitmap, the map is marked bad. The kernel considers the maps to be invalid, so does not do any more resource allocation from maps. This situation can cause the file system to report `out of space` or `out of inode` error messages even though `df` may report an adequate amount of free space.<br>This error may also occur due to bitmap inconsistencies. If a bitmap fails a consistency check, or blocks are freed that are already free in the bitmap, the file system has been corrupted. This may have occurred because a user or process wrote directly to the device or used `fsdb` to change the file system.<br>The `VX_FULLFSCK` flag is set. If the map that failed was a free extent bitmap, and the `VX_FULLFSCK` flag can't be set, then the file system is disabled.<br>◆ Action<br>Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Unmount the file system and use `fsck` to run a full structural check. |
| 006, 007 | WARNING: msgcnt *x*: mesg 006: V-2-06: vx_sumupd - *mount_point* file system summary update in au *aun* failed<br>WARNING: msgcnt *x*: mesg 007: V-2-07: vx_sumupd - *mount_point* file system summary update in inode au *iaun* failed<br>◆ Description<br>An I/O error occurred while writing the allocation unit or inode allocation unit bitmap summary to disk. This sets the `VX_FULLFSCK` flag on the file system. If the `VX_FULLFSCK` flag can't be set, the file system is disabled.<br>◆ Action<br>Check the console log for I/O errors. If the problem was caused by a disk failure, replace the disk before the file system is mounted for write access, and use `fsck` to run a full structural check. |



| Message Number | Message and Definition |
|---|---|
| 008, 009 | WARNING: msgcnt *x*: mesg 008: V-2-08: vx_direrr: *function* - *mount_point* file system dir inode *dir_inumber* dev/block *device_ID/block* dirent inode *dirent_inumber* error *errno*<br>WARNING: msgcnt *x*: mesg 009: V-2-09: vx_direrr: *function* - *mount_point* file system dir inode *dir_inumber* dirent inode *dirent_inumber* immediate directory error *errno*<br>◆ Description<br>A directory operation failed in an unexpected manner. The mount point, inode, and block number identify the failing directory. If the inode is an immediate directory, the directory entries are stored in the inode, so no block number is reported. If the error is `ENOENT` or `ENOTDIR`, an inconsistency was detected in the directory block. This inconsistency could be a bad free count, a corrupted hash chain, or any similar directory structure error. If the error is `EIO` or `ENXIO`, an I/O failure occurred while reading or writing the disk block.<br>The `VX_FULLFSCK` flag is set in the super-block so that `fsck` will do a full structural check the next time it is run.<br>◆ Action<br>Check the console log for I/O errors. If the problem was caused by a disk failure, replace the disk before the file system is mounted for write access. Unmount the file system and use `fsck` to run a full structural check. |
| 010 | WARNING: msgcnt *x*: mesg 010: V-2-10: vx_ialloc - *mount_point* file system inode *inumber* not free<br>◆ Description<br>When the kernel allocates an inode from the free inode bitmap, it checks the mode and link count of the inode. If either is non-zero, the free inode bitmap or the inode list is corrupted.<br>The `VX_FULLFSCK` flag is set in the super-block so that `fsck` will do a full structural check the next time it is run.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. |

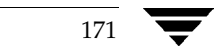
| Message Number | Message and Definition |
|---|---|
| 011 | NOTICE: msgcnt *x*: mesg 011: V-2-11: vx_noinode - *mount_point* file system out of inodes<br>◆ Description<br>The file system is out of inodes.<br>◆ Action<br>Monitor the free inodes in the file system. If the file system is getting full, create more inodes either by removing files or by expanding the file system. File system resizing is described in "Online System Administration" on page 9, and in the `fsadm_vxfs`(1M) online manual page. |
| 012 | WARNING: msgcnt *x*: mesg 012: V-2-12: vx_iget - *mount_point* file system invalid inode number *inumber*<br>◆ Description<br>When the kernel tries to read an inode, it checks the inode number against the valid range. If the inode number is out of range, the data structure that referenced the inode number is incorrect and must be fixed.<br>The `VX_FULLFSCK` flag is set in the super-block so that `fsck` will do a full structural check the next time it is run.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. |
| 013 | WARNING: msgcnt *x*: mesg 013: V-2-13: vx_iposition - *mount_point* file system inode *inumber* invalid inode list extent<br>◆ Description<br>For a Version 2 and above disk layout, the inode list is dynamically allocated. When the kernel tries to read an inode, it must look up the location of the inode in the inode list file. If the kernel finds a bad extent, the inode can't be accessed. All of the inode list extents are validated when the file system is mounted, so if the kernel finds a bad extent, the integrity of the inode list is questionable. This is a very serious error.<br>The `VX_FULLFSCK` flag is set in the super-block and the file system is disabled.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. |



| Message Number | Message and Definition |
|---|---|
| 014 | WARNING: msgcnt *x*: mesg 014: V-2-14: vx_iget - inode table overflow<br>◆ Description<br>All the system in-memory inodes are busy and an attempt was made to use a new inode.<br>◆ Action<br>Note: The tunable parameter `vx_ninode` is used to set the value of `vxfs_ninode`.<br>Look at the processes that are running and determine which processes are using inodes. If it appears there are runaway processes, they might be tying up the inodes. If the system load appears normal, increase the `vx_ninode` parameter in the kernel (see "Kernel Tunables" on page 32). |
| 015 | WARNING: msgcnt *x*: mesg 015: V-2-15: vx_ibadinactive - *mount_point* file system can't mark inode *inumber* bad<br>WARNING: msgcnt *x*: mesg 015: V-2-15: vx_ilisterr - *mount_point* file system can't mark inode *inumber* bad<br>◆ Description<br>An attempt to mark an inode bad on disk, and the super-block update to set the `VX_FULLFSCK` flag, failed. This indicates that a catastrophic disk error may have occurred since both an inode list block and the super-block had I/O failures. The file system is disabled to preserve file system integrity.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the disk failed, replace it before remounting the file system. |
| 016 | WARNING: msgcnt *x*: mesg 016: V-2-16: vx_ilisterr - *mount_point* file system error reading inode *inumber*<br>◆ Description<br>An I/O error occurred while reading the inode list. The `VX_FULLFSCK` flag is set.<br>◆ Action<br>Check the console log for I/O errors. If the problem was caused by a disk failure, replace the disk before the file system is mounted for write access. Unmount the file system and use `fsck` to run a full structural check. |

| Message Number | Message and Definition |
|---|---|
| 017 | WARNING: msgcnt *x*: mesg 017: V-2-17: vx_attr_getblk - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_attr_iget - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_attr_indadd - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_attr_indtrunc - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_attr_iremove - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_bmap - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_bmap_indirect_ext4 - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_delbuf_flush - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_dio_iovec - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_dirbread - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_dircreate - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_dirlook - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_doextop_iau - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_doextop_now - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_do_getpage - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_enter_ext4 - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_exttrunc - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_get_alloc - *mount_point* file system inode *inumber* marked bad in core |





| Message Number | Message and Definition |
|---|---|
| 017 (continued) | WARNING: msgcnt *x*: mesg 017: V-2-17: vx_ilisterr - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_indtrunc - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_iread - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_iremove - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_iremove_attr - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_logwrite_flush - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_oltmount_iget - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_overlay_bmap - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_readnomap - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_reorg_trunc - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_stablestore - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_tranitimes - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_trunc - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_write_alloc2 - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_write_default - *mount_point* file system inode *inumber* marked bad in core<br>WARNING: msgcnt *x*: mesg 017: V-2-17: vx_zero_alloc - *mount_point* file system inode *inumber* marked bad in core |

| Message Number | Message and Definition |
|---|---|
| 017 (continued) | ◆ Description<br><br>When inode information is no longer dependable, the kernel marks it bad in memory. This is followed by a message to mark it bad on disk as well unless the `mount` command `ioerror` option is set to `disable`, or there is subsequent I/O failure when updating the inode on disk. No further operations can be performed on the inode.<br><br>The most common reason for marking an inode bad is a disk I/O failure. If there is an I/O failure in the inode list, on a directory block, or an indirect address extent, the integrity of the data in the inode, or the data the kernel tried to write to the inode list, is questionable. In these cases, the disk driver prints an error message and one or more inodes are marked bad.<br><br>The kernel also marks an inode bad if it finds a bad extent address, invalid inode fields, or corruption in directory data blocks during a validation check. A validation check failure indicates the file system has been corrupted. This usually occurs because a user or process has written directly to the device or used `fsdb` to change the file system.<br><br>The `VX_FULLFSCK` flag is set in the super-block so `fsck` will do a full structural check the next time it is run.<br><br>◆ Action<br><br>Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process is writing to the device, report the problem to your customer support organization. In either case, unmount the file system. The file system can be remounted without a full `fsck` unless the `VX_FULLFSCK` flag is set for the file system. |



| Message Number | Message and Definition |
|---|---|
| 019 | WARNING: msgcnt *x*: mesg 019: V-2-19: vx_log_add - *mount_point* file system log overflow<br>◆ Description<br>Log ID overflow. When the log ID reaches `VX_MAXLOGID` (approximately one billion by default), a flag is set so the file system resets the log ID at the next opportunity. If the log ID has not been reset, when the log ID reaches `VX_DISLOGID` (approximately `VX_MAXLOGID` plus 500 million by default), the file system is disabled. Since a log reset will occur at the next 60 second sync interval, this should never happen.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. |
| 020 | WARNING: msgcnt *x*: mesg 020: V-2-20: vx_logerr - *mount_point* file system log error *errno*<br>◆ Description<br>Intent log failed. The kernel will try to set the `VX_FULLFSCK` and `VX_LOGBAD` flags in the super-block to prevent running a log replay. If the super-block can't be updated, the file system is disabled.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the disk failed, replace it before remounting the file system. |

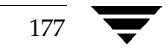
| Message Number | Message and Definition |
|---|---|
| 021 | WARNING: msgcnt *x*: mesg 021: V-2-21: vx_fs_init - *mount_point* file system validation failure<br>◆ Description<br>When a VxFS file system is mounted, the structure is read from disk. If the file system is marked clean, the structure is correct and the first block of the intent log is cleared.<br>If there is any I/O problem or the structure is inconsistent, the kernel sets the `VX_FULLFSCK` flag and the mount fails.<br>If the error isn't related to an I/O failure, this may have occurred because a user or process has written directly to the device or used `fsdb` to change the file system.<br>◆ Action<br>Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process is writing to the device, report the problem to your customer support organization. In either case, unmount the file system and use `fsck` to run a full structural check. |



| Message Number | Message and Definition |
|---|---|
| 022 | WARNING: msgcnt *x*: mesg 022: V-2-22: vx_mountroot - root file system remount failed<br>◆ Description<br>The remount of the root file system failed. The system will not be usable if the root file system can't be remounted for read/write access.<br>When a VERITAS root file system is first mounted, it is mounted for read-only access. After `fsck` is run, the file system is remounted for read/write access. The remount fails if `fsck` completed a resize operation or modified a file that was opened before the `fsck` was run. It also fails if an I/O error occurred during the remount.<br>Usually, the system halts or reboots automatically.<br>◆ Action<br>Reboot the system. The system either remounts the root cleanly or runs a full structural `fsck` and remounts cleanly. If the remount succeeds, no further action is necessary.<br>Check the console log for I/O errors. If the disk has failed, replace it before the file system is mounted for write access.<br>If the system won't come up and a full structural `fsck` hasn't been run, reboot the system on a backup root and manually run a full structural `fsck`. If the problem persists after the full structural `fsck` and there are no I/O errors, contact your customer support organization. |
| 023 | WARNING: msgcnt *x*: mesg 023: V-2-23: vx_unmountroot - root file system is busy and can't be unmounted cleanly<br>◆ Description<br>There were active files in the file system and they caused the unmount to fail.<br>When the system is halted, the root file system is unmounted. This happens occasionally when a process is hung and it can't be killed before unmounting the root.<br>◆ Action<br>`fsck` will run when the system is rebooted. It should clean up the file system. No other action is necessary.<br>If the problem occurs every time the system is halted, determine the cause and contact your customer support organization. |

| Message Number | Message and Definition |
|---|---|
| 024 | WARNING: msgcnt *x*: mesg 024: V-2-24: vx_cutwait - *mount_point* file system current usage table update error<br>◆ Description<br>Update to the current usage table (CUT) failed.<br>For a Version 2 disk layout, the CUT contains a fileset version number and total number of blocks used by each fileset.<br>The `VX_FULLFSCK` flag is set in the super-block. If the super-block can't be written, the file system is disabled.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. |
| 025 | WARNING: msgcnt *x*: mesg 025: V-2-25: vx_wsuper - *mount_point* file system super-block update failed<br>◆ Description<br>An I/O error occurred while writing the super-block during a resize operation. The file system is disabled.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the problem is a disk failure, replace the disk before the file system is mounted for write access. |
| 026 | WARNING: msgcnt *x*: mesg 026: V-2-26: vx_snap_copyblk - *mount_point* primary file system read error<br>◆ Description<br>Snapshot file system error.<br>When the primary file system is written, copies of the original data must be written to the snapshot file system. If a read error occurs on a primary file system during the copy, any snapshot file system that doesn't already have a copy of the data is out of date and must be disabled.<br>◆ Action<br>An error message for the primary file system prints. Resolve the error on the primary file system and rerun any backups or other applications that were using the snapshot that failed when the error occurred. |

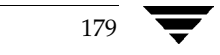




| Message Number | Message and Definition |
|---|---|
| 027 | WARNING: msgcnt *x*: mesg 027: V-2-27: vx_snap_bpcopy - *mount_point* snapshot file system write error<br>◆ Description<br>A write to the snapshot file system failed.<br>As the primary file system is updated, copies of the original data are read from the primary file system and written to the snapshot file system. If one of these writes fails, the snapshot file system is disabled.<br>◆ Action<br>Check the console log for I/O errors. If the disk has failed, replace it. Resolve the error on the disk and rerun any backups or other applications that were using the snapshot that failed when the error occurred. |
| 028 | WARNING: msgcnt *x*: mesg 028: V-2-28: vx_snap_alloc - *mount_point* snapshot file system out of space<br>◆ Description<br>The snapshot file system ran out of space to store changes.<br>During a snapshot backup, as the primary file system is modified, the original data is copied to the snapshot file system. This error can occur if the snapshot file system is left mounted by mistake, if the snapshot file system was given too little disk space, or the primary file system had an unexpected burst of activity. The snapshot file system is disabled.<br>◆ Action<br>Make sure the snapshot file system was given the correct amount of space. If it was, determine the activity level on the primary file system. If the primary file system was unusually busy, rerun the backup. If the primary file system is no busier than normal, move the backup to a time when the primary file system is relatively idle or increase the amount of disk space allocated to the snapshot file system.<br>Rerun any backups that failed when the error occurred. |

| Message Number | Message and Definition |
|---|---|
| 029, 030 | WARNING: msgcnt *x*: mesg 029: V-2-29: vx_snap_getbp - *mount_point* snapshot file system block map write error<br>WARNING: msgcnt *x*: mesg 030: V-2-30: vx_snap_getbp - *mount_point* snapshot file system block map read error<br>◆ Description<br>During a snapshot backup, each snapshot file system maintains a block map on disk. The block map tells the snapshot file system where data from the primary file system is stored in the snapshot file system. If an I/O operation to the block map fails, the snapshot file system is disabled.<br>◆ Action<br>Check the console log for I/O errors. If the disk has failed, replace it. Resolve the error on the disk and rerun any backups that failed when the error occurred. |
| 031 | WARNING: msgcnt *x*: mesg 031: V-2-31: vx_disable - *mount_point* file system disabled<br>◆ Description<br>File system disabled, preceded by a message that specifies the reason. This usually indicates a serious disk problem.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. If the problem is a disk failure, replace the disk before the file system is mounted for write access. |
| 032 | WARNING: msgcnt *x*: mesg 032: V-2-32: vx_disable - *mount_point* snapshot file system disabled<br>◆ Description<br>Snapshot file system disabled, preceded by a message that specifies the reason.<br>◆ Action<br>Unmount the snapshot file system, correct the problem specified by the message, and rerun any backups that failed due to the error. |





| Message Number | Message and Definition |
|---|---|
| 033 | WARNING: msgcnt *x*: mesg 033: V-2-33: vx_check_badblock - *mount_point* file system had an I/O error, setting VX_FULLFSCK<br>◆ Description<br>When the disk driver encounters an I/O error, it sets a flag in the super-block structure. If the flag is set, the kernel will set the `VX_FULLFSCK` flag as a precautionary measure. Since no other error has set the `VX_FULLFSCK` flag, the failure probably occurred on a data block.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the problem is a disk failure, replace the disk before the file system is mounted for write access. |
| 034 | WARNING: msgcnt *x*: mesg 034: V-2-34: vx_resetlog - *mount_point* file system can't reset log<br>◆ Description<br>The kernel encountered an error while resetting the log ID on the file system. This happens only if the super-block update or log write encountered a device failure. The file system is disabled to preserve its integrity.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the problem is a disk failure, replace the disk before the file system is mounted for write access. |
| 035 | WARNING: msgcnt *x*: mesg 035: V-2-35: vx_inactive - *mount_point* file system inactive of locked inode *inumber*<br>◆ Description<br>`VOP_INACTIVE` was called for an inode while the inode was being used. This should never happen, but if it does, the file system is disabled.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. Report as a bug to your customer support organization. |

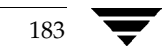
| Message Number | Message and Definition |
|---|---|
| 036 | WARNING: msgcnt *x*: mesg 036: V-2-36: vx_lctbad - *mount_point* file system link count table *lctnumber* bad<br>◆ Description<br>Update to the link count table (LCT) failed.<br>For a Version 2 and above disk layout, the LCT contains the link count for all the structural inodes. The `VX_FULLFSCK` flag is set in the super-block. If the super-block can't be written, the file system is disabled.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. |
| 037 | WARNING: msgcnt *x*: mesg 037: V-2-37: vx_metaioerr - *function* - *volume_name* file system meta data [read \| write] error in dev/block *device_ID/block*<br>◆ Description<br>A read or a write error occurred while accessing file system metadata. The full `fsck` flag on the file system was set. The message specifies whether the disk I/O that failed was a read or a write.<br>File system metadata includes inodes, directory blocks, and the file system log. If the error was a write error, it is likely that some data was lost. This message should be accompanied by another file system message describing the particular file system metadata affected, as well as a message from the disk driver containing information about the disk I/O error.<br>◆ Action<br>Resolve the condition causing the disk error. If the error was the result of a temporary condition (such as accidentally turning off a disk or a loose cable), correct the condition. Check for loose cables, etc. Unmount the file system and use `fsck` to run a full structural check (possibly with loss of data).<br>In case of an actual disk error, if it was a read error and the disk driver remaps bad sectors on write, it may be fixed when `fsck` is run since `fsck` is likely to rewrite the sector with the read error. In other cases, you replace or reformat the disk drive and restore the file system from backups. Consult the documentation specific to your system for information on how to recover from disk errors. The disk driver should have printed a message that may provide more information. |



| Message Number | Message and Definition |
|---|---|
| 038 | WARNING: msgcnt *x*: mesg 038: V-2-38: vx_dataioerr - *volume_name* file system file data [read \| write] error in dev/block *device_ID/block*<br>◆ Description<br>A read or a write error occurred while accessing file data. The message specifies whether the disk I/O that failed was a read or a write. File data includes data currently in files and free blocks. If the message is printed because of a read or write error to a file, another message that includes the inode number of the file will print. The message may be printed as the result of a read or write error to a free block, since some operations allocate an extent and immediately perform I/O to it. If the I/O fails, the extent is freed and the operation fails. The message is accompanied by a message from the disk driver regarding the disk I/O error.<br>◆ Action<br>Resolve the condition causing the disk error. If the error was the result of a temporary condition (such as accidentally turning off a disk or a loose cable), correct the condition. Check for loose cables, etc. If any file data was lost, restore the files from backups. Determine the file names from the inode number (see the `ncheck(1M)` manual page for more information).<br>If an actual disk error occurred, make a backup of the file system, replace or reformat the disk drive, and restore the file system from the backup. Consult the documentation specific to your system for information on how to recover from disk errors. The disk driver should have printed a message that may provide more information. |

| Message Number | Message and Definition |
|---|---|
| 039 | WARNING: msgcnt *x*: mesg 039: V-2-39: vx_writesuper - file system super-block write error<br>◆ Description<br>An attempt to write the file system super block failed due to a disk I/O error. If the file system was being mounted at the time, the mount will fail. If the file system was mounted at the time and the full `fsck` flag was being set, the file system will probably be disabled and Message 031 will also be printed. If the super-block was being written as a result of a `sync` operation, no other action is taken.<br>◆ Action<br>Resolve the condition causing the disk error. If the error was the result of a temporary condition (such as accidentally turning off a disk or a loose cable), correct the condition. Check for loose cables, etc. Unmount the file system and use `fsck` to run a full structural check.<br>If an actual disk error occurred, make a backup of the file system, replace or reformat the disk drive, and restore the file system from backups. Consult the documentation specific to your system for information on how to recover from disk errors. The disk driver should have printed a message that may provide more information. |
| 040 | WARNING: msgcnt *x*: mesg 040: V-2-40: vx_dqbad - *mount_point* file system user quota file update error for id *id*<br>◆ Description<br>An update to the user quotas file failed for the user ID.<br>The quotas file keeps track of the total number of blocks and inodes used by each user, and also contains soft and hard limits for each user ID. The `VX_FULLFSCK` flag is set in the super-block. If the super-block cannot be written, the file system is disabled.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the disk has a hardware failure, it should be repaired before the file system is mounted for write access. |

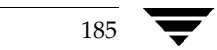




| Message Number | Message and Definition |
|---|---|
| 041 | WARNING: msgcnt *x*: mesg 041: V-2-41: vx_dqget - *mount_point* file system user quota file can't read quota for id *id*<br>◆ Description<br>A read of the user quotas file failed for the `uid`.<br>The quotas file keeps track of the total number of blocks and inodes used by each user, and contains soft and hard limits for each user ID. The `VX_FULLFSCK` flag is set in the super-block. If the super-block cannot be written, the file system is disabled.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. Check the console log for I/O errors. If the disk has a hardware failure, it should be repaired before the file system is mounted for write access. |
| 042 | WARNING: msgcnt *x*: mesg 042: V-2-42: vx_bsdquotaupdate - *mount_point* file system *user_id* disk limit reached<br>◆ Description<br>The hard limit on blocks was reached. Further attempts to allocate blocks for files owned by the user will fail.<br>◆ Action<br>Remove some files to free up space. |
| 043 | WARNING: msgcnt *x*: mesg 043: V-2-43: vx_bsdquotaupdate - *mount_point* file system *user_id* disk quota exceeded too long<br>◆ Description<br>The soft limit on blocks was exceeded continuously for longer than the soft quota time limit. Further attempts to allocate blocks for files will fail.<br>◆ Action<br>Remove some files to free up space. |

| Message Number | Message and Definition |
|---|---|
| 044 | WARNING: msgcnt *x*: mesg 044: V-2-44: vx_bsdquotaupdate - *mount_point* file system *user_id* disk quota exceeded<br>◆ Description<br>The soft limit on blocks is exceeded. Users can exceed the soft limit for a limited amount of time before allocations begin to fail. After the soft quota time limit has expired, subsequent attempts to allocate blocks for files fail.<br>◆ Action<br>Remove some files to free up space. |
| 045 | WARNING: msgcnt *x*: mesg 045: V-2-45: vx_bsdiquotaupdate - *mount_point* file system *user_id* inode limit reached<br>◆ Description<br>The hard limit on inodes was exceeded. Further attempts to create files owned by the user will fail.<br>◆ Action<br>Remove some files to free inodes. |
| 046 | WARNING: msgcnt *x*: mesg 046: V-2-46: vx_bsdiquotaupdate - *mount_point* file system *user_id* inode quota exceeded too long<br>◆ Description<br>The soft limit on inodes has been exceeded continuously for longer than the soft quota time limit. Further attempts to create files owned by the user will fail.<br>◆ Action<br>Remove some files to free inodes. |
| 047 | WARNING: msgcnt *x*: mesg 047: V-2-47: vx_bsdiquotaupdate - warning: *mount_point* file system *user_id* inode quota exceeded<br>◆ Description<br>The soft limit on inodes was exceeded. The soft limit can be exceeded for a certain amount of time before attempts to create new files begin to fail. Once the time limit has expired, further attempts to create files owned by the user will fail.<br>◆ Action<br>Remove some files to free inodes. |





| Message Number | Message and Definition |
|---|---|
| 048, 049 | WARNING: msgcnt *x*: mesg 048: V-2-48: vx_dqread - warning: *mount_point* file system external user quota file read failed<br>WARNING: msgcnt *x*: mesg 049: V-2-49: vx_dqwrite - warning: *mount_point* file system external user quota file write failed<br>◆ Description<br>To maintain reliable usage counts, VxFS maintains the user quotas files as structural files in the structural fileset. These files are updated as part of the transactions that allocate and free blocks and inodes. For compatibility with the quota administration utilities, VxFS also supports the standard user visible quota files.<br>When quotas are turned off, synced, or new limits are added, VxFS tries to update the external quota files. When quotas are enabled, VxFS tries to read the quota limits from the external quotas file. If these reads or writes fail, the external quotas file is out of date.<br>◆ Action<br>Determine the reason for the failure on the external quotas file and correct it. Recreate the quotas file. |
| 056 | WARNING: msgcnt *x*: mesg 056: V-2-56: vx_mapbad - *mount_point* file system extent allocation unit state bitmap number *number* marked bad<br>◆ Description<br>If there is an I/O failure while writing a bitmap, the map is marked bad. The kernel considers the maps to be invalid, so does not do any more resource allocation from maps. This situation can cause the file system to report "out of space" or "out of inode" error messages even though `df` may report an adequate amount of free space.<br>This error may also occur due to bitmap inconsistencies. If a bitmap fails a consistency check, or blocks are freed that are already free in the bitmap, the file system has been corrupted. This may have occurred because a user or process wrote directly to the device or used `fsdb` to change the file system.<br>The `VX_FULLFSCK` flag is set. If the `VX_FULLFSCK` flag can't be set, the file system is disabled.<br>◆ Action<br>Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Unmount the file system and use `fsck` to run a full structural check. |

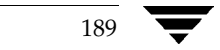
| Message Number | Message and Definition |
|---|---|
| 057 | WARNING: msgcnt *x*: mesg 057: V-2-57: vx_esum_bad - *mount_point* file system extent allocation unit summary number *number* marked bad<br>◆ Description<br>An I/O error occurred reading or writing an extent allocation unit summary.<br>The VX_FULLFSCK flag is set. If the VX_FULLFSCK flag can't be set, the file system is disabled.<br>◆ Action<br>Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Unmount the file system and use `fsck` to run a full structural check. |
| 058 | WARNING: msgcnt *x*: mesg 058: V-2-58: vx_isum_bad - *mount_point* file system inode allocation unit summary number *number* marked bad<br>◆ Description<br>An I/O error occurred reading or writing an inode allocation unit summary.<br>The VX_FULLFSCK flag is set. If the VX_FULLFSCK flag cannot be set, the file system is disabled.<br>◆ Action<br>Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Unmount the file system and use `fsck` to run a full structural check. |



| Message Number | Message and Definition |
|---|---|
| 059 | WARNING: msgcnt *x*: mesg 059: V-2-59: vx_snap_getbitbp - *mount_point* snapshot file system bitmap write error<br>◆ Description<br>An I/O error occurred while writing to the snapshot file system bitmap. There is no problem with the snapped file system, but the snapshot file system is disabled.<br>◆ Action<br>Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Restart the snapshot on an error free disk partition. Rerun any backups that failed when the error occurred. |
| 060 | WARNING: msgcnt *x*: mesg 060: V-2-60: vx_snap_getbitbp - *mount_point* snapshot file system bitmap read error<br>◆ Description<br>An I/O error occurred while reading the snapshot file system bitmap. There is no problem with snapped file system, but the snapshot file system is disabled.<br>◆ Action<br>Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process was writing to the device, report the problem to your customer support organization. Restart the snapshot on an error free disk partition. Rerun any backups that failed when the error occurred. |
| 061 | WARNING: msgcnt *x*: mesg 061: V-2-61: vx_resize - *mount_point* file system remount failed<br>◆ Description<br>During a file system resize, the remount to the new size failed. The `VX_FULLFSCK` flag is set and the file system is disabled.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. After the check, the file system shows the new size. |

| Message Number | Message and Definition |
|---|---|
| 062 | NOTICE: msgcnt *x*: mesg 062: V-2-62: vx_attr_creatop - invalid disposition returned by attribute driver<br>◆ Description<br>A registered extended attribute intervention routine returned an invalid return code to the VxFS driver during extended attribute inheritance.<br>◆ Action<br>Determine which vendor supplied the registered extended attribute intervention routine and contact their customer support organization. |
| 063 | WARNING: msgcnt *x*: mesg 063: V-2-63: vx_fset_markbad - *mount_point* file system *mount_point* fileset (index *number*) marked bad<br>◆ Description<br>An error occurred while reading or writing a fileset structure. `VX_FULLFSCK` flag is set. If the `VX_FULLFSCK` flag can't be set, the file system is disabled.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. |
| 064 | WARNING: msgcnt *x*: mesg 064: V-2-64: vx_ivalidate - *mount_point* file system inode *number* version number exceeds fileset's<br>◆ Description<br>During inode validation, a discrepancy was found between the inode version number and the fileset version number. The inode may be marked bad, or the fileset version number may be changed, depending on the ratio of the mismatched version numbers.<br>`VX_FULLFSCK` flag is set. If the `VX_FULLFSCK` flag can't be set, the file system is disabled.<br>◆ Action<br>Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process is writing to the device, report the problem to your customer support organization. In either case, unmount the file system and use `fsck` to run a full structural check. |





| Message Number | Message and Definition |
|---|---|
| 066 | NOTICE: msgcnt *x*: mesg 066: V-2-66: DMAPI mount event - *buffer*<br>◆ Description<br>An HSM (Hierarchical Storage Management) agent responded to a DMAPI mount event and returned a message in *buffer*.<br>◆ Action<br>Consult the HSM product documentation for the appropriate response to the message. |
| 067 | WARNING: msgcnt *x*: mesg 067: V-2-67: mount of *device_path* requires HSM agent<br>◆ Description<br>The file system mount failed because the file system was marked as being under the management of an HSM agent, and no HSM agent was found during the mount.<br>◆ Action<br>Restart the HSM agent and try to mount the file system again. |
| 069 | WARNING: msgcnt *x*: mesg 069: V-2-69: memory usage specified by the vxfs:vxfs_ninode and vxfs:vx_bc_bufhwm parameters exceeds available memory; the system may hang under heavy load<br>◆ Description<br>The value of the system tunable parameters—`vx_ninode` and `vx_bc_bufhwm`—add up to a value that is more than 66% of the kernel virtual address space or more than 50% of the physical system memory. VxFS inodes require approximately one kilobyte each, so both values can be treated as if they are in units of one kilobyte.<br>◆ Action<br>To avoid a system hang, reduce the value of one or both parameters to less than 50% of physical memory or to 66% of kernel virtual memory. For more information on performance and tuning, see "VxFS Performance: Creating, Mounting, and Tuning File Systems" on page 21. Note: The tunable parameter `vx_ninode` is used to set the value of `vxfs_ninode`. |

| Message Number | Message and Definition |
|---|---|
| 070 | WARNING: msgcnt *x*: mesg 070: V-2-70: checkpoint *checkpoint_name* removed from file system *mount_point*<br>◆ Description<br>The file system ran out of space while updating a Storage Checkpoint. The Storage Checkpoint was removed to allow the operation to complete.<br>◆ Action<br>Increase the size of the file system. If the file system size cannot be increased, remove files to create sufficient space for new Storage Checkpoints. Monitor capacity of the file system closely to ensure it does not run out of space. See the `fsadm_vxfs`(1M) manual page more information. |
| 071 | NOTICE: msgcnt *x*: mesg 071: V-2-71: cleared data I/O error flag in *mount_point* file system<br>◆ Description<br>The user data I/O error flag was reset when the file system was mounted. This message indicates that a read or write error occurred (see Message Number 038) while the file system was previously mounted.<br>◆ Action<br>Informational only, no action required. |
| 072 | WARNING: msgcnt *x*: vxfs: mesg 072: could not failover for *volume_name* file system<br>◆ Description<br>This message is specific to the cluster file system. The message indicates a problem in a scenario where a node failure has occurred in the cluster and the newly selected primary node encounters a failure.<br>◆ Action<br>Save the system logs and core dump of the node along with the disk image (metasave) and contact your customer support organization. The node can be rebooted to join the cluster. |



| Message Number | Message and Definition |
|---|---|
| 074 | WARNING: msgcnt *x*: mesg 074: V-2-74: vx_rcfg - *fileset_number* fileset not found while remounting *mount_point*<br>◆ Description<br>This message is specific to the cluster file system. The message indicates a problem in a scenario where a node failure has occurred in the cluster and the newly selected primary node encounters a failure.<br>◆ Action<br>Save the core dump of the node and contact your customer support organization. The node can be rebooted to join the cluster. |
| 075 | WARNING: msgcnt *x*: mesg 075: V-2-75: replay fsck failed for *mount_point* file system<br>◆ Description<br>The log replay failed during a failover or while migrating the CFS primary-ship to one of the secondary cluster nodes. The file system was disabled.<br>◆ Action<br>Unmount the file system from the cluster. Use `fsck` to run a full structural check and mount the file system again. |
| 076 | NOTICE: msgcnt *x*: mesg 076: V-2-76: checkpoint asynchronous operation on *mount_point* file system still in progress<br>◆ Description<br>An EBUSY message was received while trying to unmount a file system. The unmount failure was caused by a pending asynchronous fileset operation, such as a fileset removal or fileset conversion to a nodata Storage Checkpoint.<br>◆ Action<br>The operation may take a considerable length of time. You can do a forced unmount (see the `umount_vxfs`(1M) manual page), or simply wait for the operation to complete so file system can be unmounted cleanly. |

| Message Number | Message and Definition |
|---|---|
| 077 | WARNING: msgcnt *x*: mesg 077: V-2-77: vx_fshdchange - *mount_point* file system *number* fileset, fileset header: checksum failed<br>◆ Description<br>Disk corruption was detected while changing fileset headers. This can occur when writing a new inode allocation unit, preventing the allocation of new inodes in the fileset.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. |
| 078 | WARNING: msgcnt *x*: mesg 078: V-2-78: vx_ilealloc - *mount_point* file system *mount_point* fileset (index *number*) ilist corrupt<br>◆ Description<br>The inode list for the fileset was corrupted and the corruption was detected while allocating new inodes. The failed system call returns an `ENOSPC` error. Any subsequent inode allocations will fail unless a sufficient number of files are removed.<br>◆ Action<br>Unmount the file system and use `fsck` to run a full structural check. |



| Message Number | Message and Definition |
|---|---|
| 079 | WARNING: msgcnt *x*: mesg 017: V-2-79: vx_attr_getblk - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_attr_iget - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_attr_indadd - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_attr_indtrunc - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_attr_iremove - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_bmap - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_bmap_indirect_ext4 - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_delbuf_flush - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_dio_iovec - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_dirbread - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_dircreate - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_dirlook - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_doextop_iau - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_doextop_now - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_do_getpage - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_enter_ext4 - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_exttrunc - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_get_alloc - *mount_point* file system inode *inumber* marked bad on disk |

| Message Number | Message and Definition |
|---|---|
| 079 (continued) | WARNING: msgcnt *x*: mesg 017: V-2-79: vx_ilisterr - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_indtrunc - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_iread - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_iremove - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_iremove_attr - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_logwrite_flush - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_oltmount_iget - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_overlay_bmap - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_readnomap - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_reorg_trunc - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_stablestore - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_tranitimes - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_trunc - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_write_alloc2 - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_write_default - *mount_point* file system inode *inumber* marked bad on disk<br>WARNING: msgcnt *x*: mesg 017: V-2-79: vx_zero_alloc - *mount_point* file system inode *inumber* marked bad on disk |



| Message Number | Message and Definition |
|---|---|
| 079 (continued) | ◆ Description<br><br>When inode information is no longer dependable, the kernel marks it bad on disk. The most common reason for marking an inode bad is a disk I/O failure. If there is an I/O failure in the inode list, on a directory block, or an indirect address extent, the integrity of the data in the inode, or the data the kernel tried to write to the inode list, is questionable. In these cases, the disk driver prints an error message and one or more inodes are marked bad.<br><br>The kernel also marks an inode bad if it finds a bad extent address, invalid inode fields, or corruption in directory data blocks during a validation check. A validation check failure indicates the file system has been corrupted. This usually occurs because a user or process has written directly to the device or used `fsdb` to change the file system.<br><br>The `VX_FULLFSCK` flag is set in the super-block so `fsck` will do a full structural check the next time it is run.<br><br>◆ Action<br><br>Check the console log for I/O errors. If the problem is a disk failure, replace the disk. If the problem is not related to an I/O failure, find out how the disk became corrupted. If no user or process is writing to the device, report the problem to your customer support organization. In either case, unmount the file system and use `fsck` to run a full structural check. |
| 080 | WARNING: msgcnt *x*: vxfs: mesg 080: Disk layout versions older than Version 4 will not be supported in the next release. It is advisable to upgrade to the latest disk layout version now. See vxupgrade(1M) for information on upgrading a VxFS file system and see the VxFS Release Notes for information on disk layout support.<br><br>◆ Action<br><br>Use the `vxupgrade` command to begin upgrading file systems using older disk layouts to Version 4. Consider the following when planning disk layout upgrades:<br><br>◆ Version 2 disk layout file systems have an 8 million inode limit.<br><br>◆ Images of Version 2 disk layout file systems created by copy utilities, such as `dd` or `volcopy`, will become unusable after a disk layout upgrade. Offline conversions tools will be provided in the next VxFS feature release to aid in migrating volume-image backup copies of Version 2 disk layout file systems to a Version 4 disk layout. |

| Message Number | Message and Definition |
|---|---|
| 081 | WARNING: msgcnt *x*: mesg 081: V-2-81: possible network partition detected<br>◆ Description<br>This message displays when CFS detects a possible network partition and disables the file system locally, that is, on the node where the message appears.<br>◆ Action<br>There are one or more private network links for communication between the nodes in a cluster. At least one link must be active to maintain the integrity of the cluster. If all the links go down, after the last network link is broken, the node can no longer communicate with other nodes in the cluster.<br>Check the network connections. After verifying that the network connections is operating correctly, unmount the disabled file system and mount it again. |
| 082 | WARNING: msgcnt *x*: mesg 082: V-2-82: *volume_name* file system is on shared volume. It may get damaged if cluster is in partitioned state.<br>◆ Description<br>If a cluster node is in a partitioned state, and if the file system is on a shared VxVM volume, this volume may become corrupted by accidental access from another node in the cluster.<br>◆ Action<br>These shared disks can also be seen by nodes in a different partition, so they can inadvertently be corrupted. So the second message 082 tells that the device mentioned is on shared volume and damage can happen only if it is a real partition problem. Do not use it on any other node until the file system is unmounted from the mounted nodes. |
| 083 | WARNING: msgcnt *x*: mesg 083: V-2-83: *mount_point* file system log is not compatible with the specified intent log I/O size<br>◆ Description<br>Either the specified `mount logiosize` size is not compatible with the file system layout, or the file system is corrupted.<br>◆ Action<br>Mount the file system again without specifying the `logiosize` option, or use a `logiosize` value compatible with the intent log specified when the file system was created. If the error persists, unmount the file system and use `fsck` to run a full structural check. |



| Message Number | Message and Definition |
|---|---|
| 084 | WARNING: msgcnt *x*: mesg 084: V-2-84: in *volume_name* quota on failed during assumption. (stage *stage_number*)<br>◆ Description<br>In a cluster file system, when the primary of the file system fails, a secondary file system is chosen to assume the role of the primary. The assuming node will be able to enforce quotas after becoming the primary.<br>If the new primary is unable to enforce quotas this message will be displayed.<br>◆ Action<br>Issue the `quotaon` command from any of the nodes that have the file system mounted. |
| 085 | WARNING: msgcnt *x*: mesg 085: V-2-85: Checkpoint quota - warning: *file_system* file system fileset quota hard limit exceeded<br>◆ Description<br>The system administrator sets the quotas for checkpoints in the form of a soft limit and hard limit. This message displays when the hard limit is exceeded.<br>◆ Action<br>Delete checkpoints or increase the hard limit. |
| 086 | WARNING: msgcnt *x*: mesg 086: V-2-86: Checkpoint quota - warning: *file_system* file system fileset quota soft limit exceeded<br>◆ Description<br>The system administrator sets the quotas for checkpoints in the form of a soft limit and hard limit. This message displays when the soft limit is exceeded.<br>◆ Action<br>Delete checkpoints or increase the soft limit. This is not a mandatory action, but is recommended. |

| Message Number | Message and Definition |
|---|---|
| 087 | WARNING: msgcnt *x*: mesg 087: V-2-87: vx_dotdot_manipulate: *file_system* file system *inumber* inode *ddnumber* dotdot inode error<br>◆ Description<br>When performing an operation that changes an inode entry, if the inode is incorrect, this message will display.<br>◆ Action<br>Run a full file system check using `fsck` to correct the errors. |
| 088 | WARNING: msgcnt *x*: mesg 088: V-2-88: quotaon on *file_system* failed; limits exceed *limit*<br>◆ Description<br>The external quota file, `quotas`, contains the quota values, which range from 0 up to 2147483647. When quotas are turned on by the `quotaon` command, this message displays when a user exceeds the quota limit.<br>◆ Action<br>Correct the quota values in the `quotas` file. |
| 089 | WARNING: msgcnt *x*: mesg 089: V-2-89: quotaon on *file_system* invalid; disk usage for *group/user* id *uid* exceeds *sectors* sectors<br>◆ Description<br>The supported quota limit is up to 2147483647 sectors. When quotas are turned on by the `quotaon` command, this message displays when a user exceeds the supported quota limit.<br>◆ Action<br>Ask the user to delete files to lower the quota below the limit. |
| 090 | WARNING: msgcnt *x*: mesg 090: V-2-90: quota on *file_system* failed; soft limits greater than hard limits<br>◆ Description<br>One or more users or groups has a soft limit set greater than the hard limit, preventing the BSD quota from being turned on.<br>◆ Action<br>Check the soft limit and hard limit for every user and group and confirm that the soft limit is not set greater than the hard limit. |



| Message Number | Message and Definition |
|---|---|
| 091 | WARNING: msgcnt *x*: mesg 091: V-2-91: vx_fcl_truncate - failure to punch hole at offset *offset* for *bytes* bytes in File Change Log file; error *error_number*<br>◆ Description<br>The vxfs kernel has experienced an error while trying to manage the space consumed by the File Change Log file. Because the space cannot be actively managed at this time, the FCL has been deactivated and has been truncated to 1 file system block, which contains the FCL superblock.<br>◆ Action<br>Re-activate the FCL. |
| 092 | WARNING: msgcnt *x*: mesg 092: V-2-92: vx_mkfcltran - failure to map offset *offset* in File Change Log file<br>◆ Description<br>The vxfs kernel was unable to map actual storage to the next offset in the File Change Log file. This is mostly likely caused by a problem with allocating to the FCL file. Because no new FCL records can be written to the FCL file, the FCL has been deactivated.<br>◆ Action<br>Re-activate the FCL. |
| 093 | WARNING: msgcnt *x*: mesg 093: V-2-93: Disk layout versions older than Version 6 will not be supported for shared mounts in the next release. It is advisable to upgrade to the latest layout version now. See vxupgrade(1M) for information on upgrading a VxFS file system and see the VxFS Release Notes for information on disk layout support.<br>◆ Action<br>Upgrade your disk layout to Version 6 for shared mounts. This is not a mandatory action, but is recommended. Disk layout Versions 4 and 5 will still be supported for local mounts in the next release of the VxFS.<br>Use the `vxupgrade` command to begin upgrading file systems using older disk layouts to Version 6. |

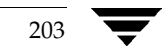
| Message Number | Message and Definition |
|---|---|
| 095 | WARNING: msgcnt *x*: mesg 095: V-2-95: Setting `vxfs_ifree_timelag` to *time* since the specified value for `vxfs_ifree_timelag` is less than the recommended minimum value of *time*.<br>◆ Description<br>The value for `vxfs_ifree_timelag` specified by the system administrator is less than the recommended minimum value, *time*, and so the value of `vxfs_ifree_timelag` has been automatically changed to *time*.<br>◆ Action<br>No corrective action required on the file system. |
| 096 | WARNING: msgcnt *x*: mesg 096: V-2-96: *file_system* file system `fullfsck` flag set - *function_name*.<br>◆ Description<br>The next time the file system is mounted, a full `fsck` must be performed.<br>◆ Action<br>No immediate action required. When the file system is unmounted, run a full file system check using `fsck` before mounting it again. |
| 097 | WARNING: msgcnt *x*: mesg 097: V-2-97: VxFS failed to create new thread (*error_number*, *function_address*:*argument_address*)<br>◆ Description<br>VxFS failed to create a kernel thread due to resource constraints, which is often a memory shortage.<br>◆ Action<br>VxFS will retry the thread creation until it succeeds; no immediate action is required. Kernel resources, such as kernel memory, might be overcommitted. If so, reconfigure the system accordingly. |



| Message Number | Message and Definition |
|---|---|
| 098 | WARNING: msgcnt *x*: mesg 098: V-2-98: VxFS failed to initialize File Change Log for *fileset* fileset (index *number*) of *mount_point* file system<br>◆ Description<br>VxFS mount failed to initialize FCL structures for the current fileset mount. As a result, FCL could not be turned on. The FCL file will have no logging records.<br>◆ Action<br>Reactivate the FCL. |
| 099 | WARNING: msgcnt *x*: mesg 099: V-2-99: The specified value for vx_ninode is less than the recommended minimum value of *min_value*<br>◆ Description<br>Auto-tuning or the value specified by the system administrator resulted in a value lower than the recommended minimum for the total number of inodes that can be present in the inode cache. VxFS will ignore the newly tuned value and will keep the value specified in the message (VX_MINNINODE).<br>◆ Action<br>Informational only; no action required. |
| 100 | WARNING: msgcnt *x*: mesg 100: V-2-100: Inode *inumber* can not be accessed: file size exceeds OS limitations.<br>◆ Description<br>The specified inode's size is larger than the file size limit of the current operating system. The file cannot be opened on the current platform. This can happen when a file is created on one OS and the filesystem is then moved to a machine running an OS with a smaller file size limit.<br>◆ Action<br>If the file system is moved to the platform on which the file was created, the file can be accessed from there. It can then be converted to multiple smaller files in a manner appropriate to the application and the file's format, or simply be deleted if it is no longer required. |

| Message Number | Message and Definition |
|---|---|
| 101 | WARNING: msgcnt *x*: mesg 101: V-2-101: File Change Log on *mount_point* for file set *index* approaching max file size supported. File Change Log will be reactivated when its size hits max file size supported.<br>◆ Description<br>The size of the FCL file is approching the maximum file size supported. This size is platform specific. When the FCL file is reaches the maximum file size, the FCL will be deactivated and reactivated. All logging information gathered so far will be lost.<br>◆ Action<br>Take any corrective action possible to restrict the loss due to the FCL being deactivated and reactivated. |
| 102 | WARNING: msgcnt *x*: mesg 102: V-2-102: File Change Log of *mount_point* for file set *index* has been reactivated.<br>◆ Description<br>The size of FCL file reached the maximum supported file size and the FCL has been reactivated. All records stored in the FCL file, starting from the current *fc_loff* up to the maximum file size, have been purged. New records will be recorded in the FCL file starting from offset *fs_bsize*. The activation time in the FCL is reset to the time of reactivation. The impact is equivalent to File Change Log being deactivated and activated.<br>◆ Action<br>Informational only; no action required. |





# Disk Layout C

The disk layout is the way file system information is stored on disk. On VxFS, six different disk layout versions were created to take advantage of evolving technological developments. The disk layout versions used on VxFS were:

| | | |
|---|---|---|
| Version 1 | The Version 1 disk layout is the original VxFS disk layout provided with pre-2.0 versions of VxFS. | Not Supported |
| Version 2 | The Version 2 disk layout supports features such as filesets, dynamic inode allocation, and enhanced security. The Version 2 layout is available with and without quotas support. | Not Supported |
| Version 3 | The Version 3 disk layout encompasses all file system structural information in files, rather than at fixed locations on disk, allowing for greater scalability. Version 3 supports files and file systems up to one terabyte in size. | Not Supported |
| Version 4 | The Version 4 disk layout encompasses all file system structural information in files, rather than at fixed locations on disk, allowing for greater scalability. Version 4 supports files and file systems up to two terabytes in size. | Supported |
| Version 5 | Version 5 enables the creation of file system sizes up to 32 terabytes. Files can be a maximum of two terabytes. File systems larger than 2 TB must be created on a VERITAS Volume Manager volume. Version 5 also enables setting up to 1024 access control list (ACL) entries. | Supported |
| Version 6 | The Version 6 disk layout enables features such as multi-volume support, cross-platform data sharing, named data streams, and file change log. | Supported |

Some of the disk layout versions were not supported on all UNIX operating systems. Version 2 and 3 file systems can still be mounted, but this will be disallowed in future releases. Currently, the Version 4, Version 5, and Version 6 disk layouts can be created and mounted. Version 6 is the default disk layout version.

The `vxupgrade` command is provided to upgrade an existing VxFS file system to the Version 4, Version 5, or Version 6 disk layout while the file system remains online. You must do an upgrade in steps from older to newer layouts. See the `vxupgrade`(1M) manual page for details on upgrading VxFS file systems.



The `vxfsconvert` command is provided to upgrade Version 2 and 3 disk layouts to the Version 4 disk layout while the file system is not mounted. Using vxfsconvert, the file system can be converted to the Version 4 layout while offline, then using `vxupgrade`, you can convert it to Version 5 while online. See the `vxfsconvert`(1M) manual page for details on upgrading VxFS disk layouts.

The following additional topics are covered in this appendix:



# Disk Space Allocation

Disk space is allocated by the system in 1024-byte sectors. An integral number of sectors are grouped together to form a logical block. VxFS supports logical block sizes of 1024, 2048, 4096, and 8192 bytes. The default block size is 1024 bytes. The block size may be specified as an argument to the `mkfs` utility and may vary between VxFS file systems mounted on the same system. VxFS allocates disk space to files in extents. An extent is a set of contiguous blocks.

# VxFS Version 4 Disk Layout

The Version 4 disk layout allows the file system to scale easily to accommodate large files and large file systems.

The original disk layouts divided up the file system space into allocation units. The first AU started part way into the file system which caused potential alignment problems depending on where the first AU started. Each allocation unit also had its own summary, bitmaps, and data blocks. Because this AU structural information was stored at the start of each AU, this also limited the maximum size of an extent that could be allocated. By replacing the allocation unit model of previous versions, the need for alignment of allocation units and the restriction on extent sizes was removed.

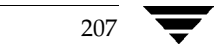
The VxFS Version 4 disk layout divides the entire file system space into fixed size allocation units. The first allocation unit starts at block zero and all allocation units are a fixed length of 32K blocks. (An exception may be the last AU, which occupies whatever space remains at the end of the file system). Because the first AU starts at block zero instead of part way through the file system as in previous versions, there is no longer a need for explicit AU alignment or padding to be added when creating a file system.

The Version 4 file system also moves away from the model of storing AU structural data at the start of an AU and puts all structural information in files. So expanding the file system structures simply requires extending the appropriate structural files. This removes the extent size restriction imposed by the previous layouts.

All Version 4 structural files reside in the *structural fileset*. The structural files in the Version 4 disk layout are:

| | |
|---|---|
| object location table file | Contains the object location table (OLT). The OLT, which is referenced from the super-block, is used to locate the other structural files. |
| label file | Encapsulates the super-block and super-block replicas. Although the location of the primary super-block is known, the label file can be used to locate super-block copies if there is structural damage to the file system. |
| device file | Records device information such as volume length and volume label, and contains pointers to other structural files. |
| fileset header file | Holds information on a per-fileset basis. This may include the inode of the fileset's inode list file, the maximum number of inodes allowed, an indication of whether the file system supports large files, and the inode number of the quotas file if the fileset supports quotas. When a file system is created, there are two filesets—the *structural fileset* defines the file system structure, the *primary fileset* contains user data. |
| inode list file | Both the primary fileset and the structural fileset have their own set of inodes stored in an inode list file. Only the inodes in the primary fileset are visible to users. When the number of inodes is increased, the kernel increases the size of the inode list file. |
| inode allocation unit file | Holds the free inode map, extended operations map, and a summary of inode resources. |
| log file | Maps the block used by the file system intent log. |
| extent allocation unit state file | Indicates the allocation state of each AU by defining whether each AU is free, allocated as a whole (no bitmaps allocated), or expanded, in which case the bitmaps associated with each AU determine which extents are allocated. |
| extent allocation unit summary file | Contains the AU summary for each allocation unit, which contains the number of free extents of each size. The summary for an extent is created only when an allocation unit is expanded for use. |
| free extent map file | Contains the free extent maps for each of the allocation units. |





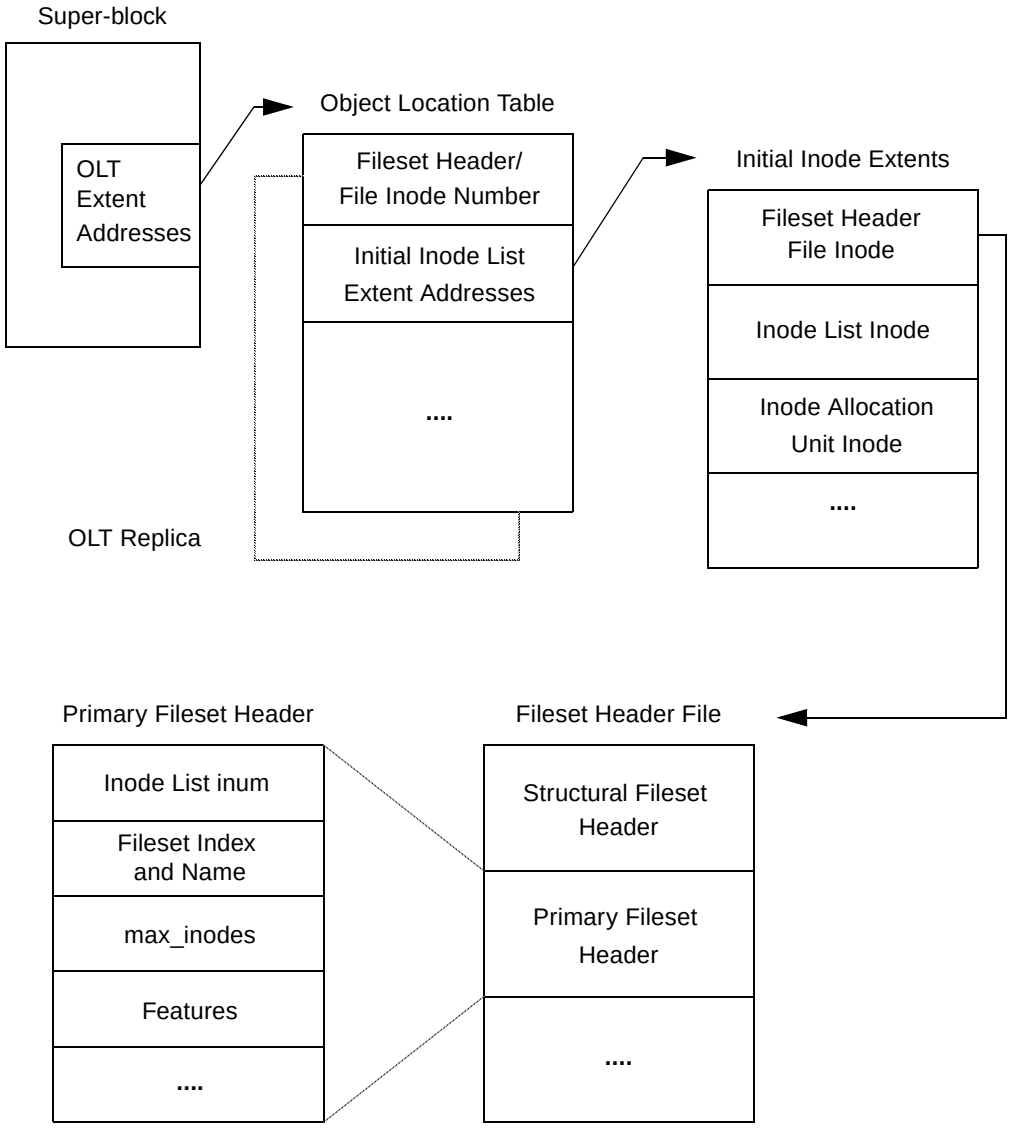
quotas files — There is a `quotas` file which is used to track the resources allocated to each user and a `quotas.grp` file to track the resources allocated to each group.

The following figure shows how the kernel and utilities build information about the structure of the file system. The super-block location is in a known location from which the OLT can be located. From the OLT, the initial extents of the structural inode list can be located along with the inode number of the fileset header file. The initial inode list extents contain the inode for the fileset header file from which the extents associated with the fileset header file are obtained.

As an example, when mounting the file system, the kernel needs to access the primary fileset in order to access its inode list, inode allocation unit, quotas file and so on. The required information is obtained by accessing the fileset header file from which the kernel can locate the appropriate entry in the file and access the required information.

The Version 4 disk layout supports Access Control Lists and Block-Level Incremental (BLI) Backup. BLI Backup is a backup method that stores and retrieves only the data blocks changed since the previous backup, not entire files. This saves times, storage space, and computing resources required to backup large databases. This file system technology is implemented in other VERITAS products. For information on how to use this feature, contact your sales channel.

VxFS Version 4 Disk Layout



# VxFS Version 5 Disk Layout

VxFS disk layout Version 5 is similar to Version 4. Structural files in Version 5 are the same in Version 4. However, the Version 5 disk layout supports file systems up to 32 terabytes. For a file system to take advantage of VxFS 32-terabyte support, it must be created on a VERITAS Volume Manager volume, and only on a 64-bit kernel operating system. The maximum file system size on a 32-bit kernel is still one terabyte. Files cannot exceed two terabytes in size. For 64-bit kernels, the maximum size of the file system you can create depends on the block size:

| Block Size | Maximum File System Size |
|---|---|
| 1024 bytes | 4,294,967,039 sectors (≈ 4 TB) |
| 2048 bytes | 8,589,934,078 sectors (≈ 8 TB) |
| 4096 bytes | 17,179,868,156 sectors (≈ 16 TB) |
| 8192 bytes | 34,359,736,312 sectors (≈ 32 TB) |

If you specify the file system size when creating a file system, the block size defaults to the appropriate value as shown above (see the `mkfs`(1M) manual page for more information).

The Version 5 disk layout also supports group quotas (see "Quota Files on VxFS" on page 98). Quota limits cannot exceed one terabyte.

The Version 5 disk layout also supports up to 1024 access control list entries.

# VxFS Version 6 Disk Layout

VxFS disk layout Version 6 is similar to Version 5. Structural files in Version 6 are the same in Version 5. The Version 6 disk layout can theoretically support files and file systems up to 8 exabytes ($2^{63}$). For a file system to take advantage of VxFS 8-exabyte support, it must be created on a VERITAS Volume Manager volume, and only on a 64-bit kernel operating system. The maximum file system size on a 32-bit kernel is still one terabyte. For 64-bit kernels, the maximum size of the file system you can create depends on the block size:

| Block Size | Currently-Supported Maximum File System Size |
|---|---|
| 1024 bytes | 68,719,472,624 sectors (≈ 32 TB) |
| 2048 bytes | 137,438,945,248 sectors (≈ 64 TB) |
| 4096 bytes | 274,877,890,496 sectors (≈ 128 TB) |
| 8192 bytes | 549,755,780,992 sectors (≈ 256 TB) |
| Note: Sector size in bytes specified by the DEV_BSIZE system parameter. | |

If you specify the file system size when creating a file system, the block size defaults to the appropriate value as shown above. See the mkfs(1M) manual page for more information.

The Version 6 disk layout also supports group quotas (see "Quota Files on VxFS" on page 98).



# Glossary

**access control list (ACL)**

The information that identifies specific users or groups and their access privileges for a particular file or directory.

**agent**

A process that manages predefined VERITAS Cluster Server (VCS) resource types. Agents bring resources online, take resources offline, and monitor resources to report any state changes to VCS. When an agent is started, it obtains configuration information from VCS and periodically monitors the resources and updates VCS with the resource status.

**allocation unit**

A group of consecutive blocks on a file system that contain resource summaries, free resource maps, and data blocks. Allocation units also contain copies of the super-block.

**API**

Application Programming Interface.

**asynchronous writes**

A delayed write in which the data is written to a page in the system's page cache, but is not written to disk before the write returns to the caller. This improves performance, but carries the risk of data loss if the system crashes before the data is flushed to disk.

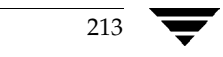
**atomic operation**

An operation that either succeeds completely or fails and leaves everything as it was before the operation was started. If the operation succeeds, all aspects of the operation take effect at once and the intermediate states of change are invisible. If any aspect of the operation fails, then the operation aborts without leaving partial changes.

**Block-Level Incremental Backup (BLI Backup)**

A VERITAS backup capability that does not store and retrieve entire files. Instead, only the data blocks that have changed since the previous backup are backed up.





### buffered I/O

During a read or write operation, data usually goes through an intermediate kernel buffer before being copied between the user buffer and disk. If the same data is repeatedly read or written, this kernel buffer acts as a cache, which can improve performance. See *unbuffered I/O* and *direct I/O*.

### CFS

VERITAS Cluster File System.

### cluster mounted file system

A shared file system that enables multiple hosts to mount and perform file operations on the same file. A cluster mount requires a shared storage device that can be accessed by other cluster mounts of the same file system. Writes to the shared device can be done concurrently from any host on which the cluster file system is mounted. To be a cluster mount, a file system must be mounted using the `mount -o cluster` option. See *local mounted file system*.

### contiguous file

A file in which data blocks are physically adjacent on the underlying media.

### CVM

The cluster functionality of VERITAS Volume Manager.

### data block

A block that contains the actual data belonging to files and directories.

### data synchronous writes

A form of synchronous I/O that writes the file data to disk before the write returns, but only marks the inode for later update. If the file size changes, the inode will be written before the write returns. In this mode, the file data is guaranteed to be on the disk before the write returns, but the inode modification times may be lost if the system crashes.

### defragmentation

The process of reorganizing data on disk by making file data blocks physically adjacent to reduce access times.

### direct extent

An extent that is referenced directly by an inode.

**direct I/O**

An unbuffered form of I/O that bypasses the kernel's buffering of data. With direct I/O, the file system transfers data directly between the disk and the user-supplied buffer. See *buffered I/O* and *unbuffered I/O*.

**discovered direct I/O**

Discovered Direct I/O behavior is similar to direct I/O and has the same alignment constraints, except writes that allocate storage or extend the file size do not require writing the inode changes before returning to the application.

**encapsulation**

A process that converts existing partitions on a specified disk to volumes. If any partitions contain file systems, `/etc/fstab` entries are modified so that the file systems are mounted on volumes instead. Encapsulation is not applicable on some systems.

**extent**

A group of contiguous file system data blocks treated as a single unit. An extent is defined by the address of the starting block and a length.

**extent attribute**

A policy that determines how a file allocates extents.

**external quotas file**

A quotas file (named `quotas`) must exist in the root directory of a file system for quota-related commands to work. See *quotas file* and *internal quotas file*.

**file system block**

The fundamental minimum size of allocation in a file system. This is equivalent to the fragment size on some UNIX file systems.

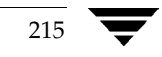
**fileset**

A collection of files within a file system.

**fixed extent size**

An extent attribute used to override the default allocation policy of the file system and set all allocations for a file to a specific fixed size.





**fragmentation**

The on-going process on an active file system in which the file system is spread further and further along the disk, leaving unused gaps or *fragments* between areas that are in use. This leads to degraded performance because the file system has fewer options when assigning a file to an extent.

**GB**

Gigabyte ($2^{30}$ bytes or 1024 megabytes).

**hard limit**

The hard limit is an absolute limit on system resources for individual users for file and data block usage on a file system. See *quota*.

**indirect address extent**

An extent that contains references to other extents, as opposed to file data itself. A *single* indirect address extent references indirect data extents. A *double* indirect address extent references single indirect address extents.

**indirect data extent**

An extent that contains file data and is referenced via an indirect address extent.

**inode**

A unique identifier for each file within a file system that contains the data and metadata associated with that file.

**inode allocation unit**

A group of consecutive blocks containing inode allocation information for a given fileset. This information is in the form of a resource summary and a free inode map.

**intent logging**

A method of recording pending changes to the file system structure. These changes are recorded in a circular *intent log* file.

**internal quotas file**

VxFS maintains an internal quotas file for its internal usage. The internal quotas file maintains counts of blocks and indices used by each user. See *quotas* and *external quotas file*.

**K**

Kilobyte ($2^{10}$ bytes or 1024 bytes).

**large file**

A file larger than two terabytes. VxFS supports files up to 8 exabytes in size.

**large file system**

A file system larger than two terabytes. VxFS supports file systems up to 8 exabytes in size.

**latency**

For file systems, this typically refers to the amount of time it takes a given file system operation to return to the user.

**local mounted file system**

A file system mounted on a single host. The single host mediates all file system writes to storage from other clients. To be a local mount, a file system cannot be mounted using the `mount -o cluster` option. See *cluster mounted file system*.

**metadata**

Structural data describing the attributes of files on a disk.

**MB**

Megabyte ($2^{20}$ bytes or 1024 kilobytes).

**mirror**

A duplicate copy of a volume and the data therein (in the form of an ordered collection of subdisks). Each mirror is one copy of the volume with which the mirror is associated.

**multi-volume file system**

A single file system that has been created over multiple volumes, with each volume having its own properties.

**MVS**

Multi-volume support.

**node**

One of the hosts in a cluster.

**node abort**

A situation where a node leaves a cluster (on an emergency basis) without attempting to stop ongoing operations.



**node join**

The process through which a node joins a cluster and gains access to shared disks.

**object location table (OLT)**

The information needed to locate important file system structural elements. The OLT is written to a fixed location on the underlying media (or disk).

**object location table replica**

A copy of the OLT in case of data corruption. The OLT replica is written to a fixed location on the underlying media (or disk).

**page file**

A fixed-size block of virtual address space that can be mapped onto any of the physical addresses available on a system.

**preallocation**

A method of allowing an application to guarantee that a specified amount of space is available for a file, even if the file system is otherwise out of space.

**primary fileset**

The files that are visible and accessible to the user.

**Quick I/O file**

A regular VxFS file that is accessed using the `::cdev:vxfs:` extension.

**Quick I/O for Databases**

Quick I/O is a VERITAS File System feature that improves database performance by minimizing read/write locking and eliminating double buffering of data. This allows online transactions to be processed at speeds equivalent to that of using raw disk devices, while keeping the administrative benefits of file systems.

**quotas**

Quota limits on system resources for individual users for file and data block usage on a file system. See *hard limit* and *soft limit*.

**quotas file**

The quotas commands read and write the external quotas file to get or change usage limits. When quotas are turned on, the quota limits are copied from the external quotas file to the internal quotas file. See *quotas*, *internal quotas file*, and *external quotas file*.

**reservation**

An extent attribute used to preallocate space for a file.

**root disk group**

A special private disk group that always exists on the system. The root disk group is named `rootdg`.

**shared disk group**

A disk group in which the disks are shared by multiple hosts (also referred to as a cluster-shareable disk group).

**shared volume**

A volume that belongs to a shared disk group and is open on more than one node at the same time.

**snapshot file system**

An exact copy of a mounted file system at a specific point in time. Used to do online backups.

**snapped file system**

A file system whose exact image has been used to create a snapshot file system.

**soft limit**

The soft limit is lower than a hard limit. The soft limit can be exceeded for a limited time. There are separate time limits for files and blocks. See *hard limit* and *quotas*.

**Storage Checkpoint**

A facility that provides a consistent and stable view of a file system or database image and keeps track of modified data blocks since the last Storage Checkpoint.

**structural fileset**

The files that define the structure of the file system. These files are not visible or accessible to the user.

**super-block**

A block containing critical information about the file system such as the file system type, layout, and size. The VxFS super-block is always located 8192 bytes from the beginning of the file system and is 8192 bytes long.



**synchronous writes**

A form of synchronous I/O that writes the file data to disk, updates the inode times, and writes the updated inode to disk. When the write returns to the caller, both the data and the inode have been written to disk.

**TB**

Terabyte ($2^{40}$ bytes or 1024 gigabytes).

**transaction**

Updates to the file system structure that are grouped together to ensure they are all completed.

**throughput**

For file systems, this typically refers to the number of I/O operations in a given unit of time.

**unbuffered I/O**

I/O that bypasses the kernel cache to increase I/O performance. This is similar to direct I/O, except when a file is extended; for direct I/O, the inode is written to disk synchronously, for unbuffered I/O, the inode update is delayed. See *buffered I/O* and *direct I/O*.

**volume**

A virtual disk which represents an addressable range of disk blocks used by applications such as file systems or databases.

**volume set**

A container for multiple different volumes. Each volume can have its own geometry.

**vxfs**

The VERITAS File System type. Used as a parameter in some commands.

**VxFS**

The VERITAS File System.

**VxVM**

The VERITAS Volume Manager.

# Index