Symantec Yellow Books™

Using Dynamic Storage Tiering

How the Veritas Storage Foundation helps you get the most from multi-tier online storage

Ganesh Karche
Murthy Mamidi
Paul Massiglia

symantec™

Symantec Corporation 20330 Stevens Creek Blvd. Cupertino, CA 95014 USA

http://www.symantec.com

ISBN 0-321-446100

Acknowledgments

Trying to describe a technology to potential users while it is still evolving is particularly challenging, especially when the evolution is occurring across continents. We are grateful to Prasanta Dash, Samir Desai, Scott Kaiser, Manish Parekh, Chuck Silvers, and Triet Vo of Symantec's Storage and Server Management Group for keeping us on track as the Dynamic Storage Tiering facility and this book developed concurrently.

Our goal in writing this book has been to describe a completely new and unfamiliar storage management capability to storage architects and administrators in the large enterprises that stand to benefit from it. To that end, our thanks go to Chris Naddeo, then of the sales engineering organization for his incisive reviews of the manuscript from his unique user-oriented perspective.

Thanks also to Chris Santilli, of Copan Systems Corporation for opening our eyes to the potential benefits of massive arrays of idle disk (MAID) technology, and its unique benefits when combined with the Storage Foundation Dynamic Storage Tiering facility. Finally, we are grateful to Marianne Lent of the SSMG Technology Strategy Office for her painstaking critique of the final manuscript and her lessons in how to speak to the priesthood of UNIX administrators.

As anyone who has done it knows, the content of a book is only part of the challenge. Sarah Connell-Smith of the Symantec Infodev organization exhibited remarkable patience and forebearance in editing a manuscript that was growing chaotically eight time zones away from her office. To help us make the words in this book clearer, we have included a number of graphics, and for these we would like to thank LMA Associates, who made our rough sketches into the finished article with great skill. Finally, our thanks to our project manager Kristine Mitchell, whose persistence and good humor as she "worked the system" to get this book from manuscript to finished product were responsible in large measure for it finding its way into your hands. If despite the extraordinary efforts of this team, errors and omissions remain, responsibility for them lies solely with the authors.

Ganesh Karche
Murthy Mamidi
Paul Massiglia

Mountain View, CA
April, 2006

Contents

Chapter 1

The challenge in using multi-tier storage effectively

This chapter includes the following topics:

About this book

This Symantec Yellow Book™, *Using Dynamic Storage Tiering*, examines the question of how enterprises can effectively exploit available technology and configuration options to substantially reduce the cost of online storage for the Linux®, Solaris™, HP-UX™, and AIX® systems in their data centers without incurring offsetting administrative cost. The book addresses the needs of data centers with upwards of a terabyte of online data facility in one or more data centers. It describes the multi-tier storage value proposition, the Storage

Foundation solution, the Dynamic Storage Tiering (DST) facility, and provides best practices, recommendations for use, and detailed technical guidelines.

Following this introduction, the main body of the book is presented in three parts. Part I, consisting of Chapters 2 through 6, describes the Veritas multi-volume file system and DST technologies. Part II, consisting of Chapters 7 through 12, describes how DST can be used to solve common enterprise data management problems. Part III includes appendixes that contain the XML document type descriptor (DTD) for DST, as well as comprehensive policy examples.

The book is intended for data center architects and administrators responsible for designing and implementing online digital data storage strategies for their enterprises' data centers, Symantec business partners who deploy storage solutions for their customers, and Symantec employees who support both groups.

Read *Using Dynamic Storage Tiering* to learn more about:

- The value proposition for multiple tiers of online storage
- Barriers to deriving benefit from deploying multiple storage tiers
- The capabilities of Storage Foundation multi-volume file systems and the Dynamic Storage Tiering facility that manages them
- How Dynamic Storage Tiering automates the placement and relocation of files in multi-volume file systems
- Common problems in the management of data center storage that are solved by Dynamic Storage Tiering
- Best practices for implementing a multi-tier storage strategy
- Sample policies for automating data placement in a multi-tier storage installation

Multi-tier storage economics and mechanics

Multi-tier storage—the use of virtual or physical storage devices with different I/O performance, data availability, and relative cost characteristics to provide differentiated online storage for computer systems—is currently a much-discussed storage management technique, both in the press and among enterprise users.

The introduction and aggressive promotion of Serial ATA (SATA)-based disk arrays has precipitated user interest in multi-tier storage. But multiple tiers of storage have been available to users since there have been disk arrays and volume managers. Table 1-1 lists three ways in which the cost of usable online storage can differ, even if the same underlying device technology is used to implement it.

Table 1-1 Cost factors in the online storage hierarchy

Cost factor	Differences
Configuration	Even when configured using identical physical storage devices, a mirrored LUN presented by a disk array or a mirrored virtual volume costs more per usable byte than a RAID-5 or non-redundant one.
Age of hardware	A volume or LUN configured using older, lower-capacity disk drives typically costs more per usable byte than one of equivalent usable capacity and availability configured from newer, higher-capacity devices of the same performance class.
Type of hardware	LUNs presented by enterprise-class disk arrays typically have a higher hardware cost per usable byte than functionally identical volumes constructed from directly connected disks of equivalent capacity and performance.

The introduction of SATA-based arrays may actually have been a catalyst rather than an enabler for multi-tier storage, heightening user awareness of the many online data storage options for their applications, each with its own availability and performance characteristics and associated cost.

Software-based virtualization technology in the form of volume managers implements multi-tier storage in the form of virtual volumes configured from the hardware devices at hand. Multiple tiers of storage hardware simply add to the number of options available to users—any type of software-based virtual volume can be created from any type of storage device or LUN presented by disk arrays.

Storage tiers

Information technology professionals tend to think of multi-tier online storage as a one-dimensional hierarchy—better quality storage costs more. To some extent this is accurate, but it is an oversimplification. In reality, the hierarchy is multi-dimensional. For example, mirrored, RAID-5, and striped LUNs or volumes of equivalent capacity form an availability hierarchy, with higher availability costing more. But different mirrored, RAID-5, and striped volumes may have different widths, or numbers of columns, resulting in different performance characteristics (and minor differences in failure protection), so there is a performance dimension to multi-tier storage as well.

Overlaid on this availability and performance hierarchy are storage hardware technology differences—volumes configured from enterprise disk array LUNs, mid-range disk array LUNs, and disks connected directly to their host systems. Even more complex configurations, such as mirrored volumes configured from

RAID-5 LUNs presented by two disk arrays, further complicate the multi-tier storage picture.

Moreover, many enterprises create storage hierarchies for non-technical reasons. For example, databases of business transactions or human resources records may be segregated from files containing engineering drawings for security reasons. The organization that owns a set of data, the applications that process it, and the I/O patterns to which it is typically subjected are also sometimes used as criteria for creating different storage tiers on which different types of data are placed.

All of this leads to the conclusion that there is more to multi-tier storage than simply an availability-based or performance-based differential cost hierarchy. Storage tiers may also be enterprise-specific, data center-specific, and potentially even application-specific. Any storage management facility that incorporates mechanisms for dealing with multi-tier storage must be sufficiently general to allow enterprise users to define 'hierarchies' that meet their particular business needs.

Requirements for multi-tier storage

Even if all storage devices cost the same and performed identically, there would be reasons to maintain and manage multiple tiers of storage.

Controlling the placement of data objects within two or more tiers of storage can be useful to enterprises for five principal reasons listed in Table 1-2.

Table 1-2 Reasons for implementing multi-tier storage

Reason	Description
Expected access patterns	Different types of data have different I/O performance needs. High data transfer performance is important for streams, but moderate I/O request rates are acceptable. By contrast, high I/O request rates are essential for most transactional data.
Load balancing	Applications that run concurrently may compete unnecessarily for I/O resources unless their data is placed on separate storage devices with separate access paths.
Availability	Different data sets have different availability requirements. Typically, enterprises can conduct business if their human resources systems are down for a day or two, but not if their point-of-sale or customer relationship management systems are down.

Table 1-2 Reasons for implementing multi-tier storage *(continued)*

Reason	Description
Data protection	Different data sets have different values. Losing a day's business transactions is significant to an enterprise, but probably survivable. Losing quarterly or annual closing figures, on the other hand, might be catastrophic. Losing an entire day's work would be a significant setback for an animator, but losing the finished product would be a much more serious setback for the animation company.
General business	Enterprise accounting, security, and regulatory compliance policies may require that certain storage devices be restricted to containing specific files or types of files.

Differences among storage devices

Storage I/O performance is influenced by both hardware cost and configuration. More expensive devices typically outperform less expensive ones. But it is also true that virtual volumes with more columns (separate disks or LUNs) tend to outperform volumes with fewer columns, and the cost per usable byte of the two is equal.

In contrast, data reliability and availability always have a strong cost component. For example, mirrored volumes cost about twice as much per usable byte as non-redundant ones; triply mirrored volumes cost three times as much, and so forth.

Storage tiers created from physically similar components may also have significantly different costs based on the functionality they provide. For example, a tier of storage may be configured to take periodic full-size snapshots or to replicate itself remotely, while another utilizing the same basic hardware components is not. Intuitively, only files whose business value justifies keeping multiple copies readily available should be stored on a (more expensive) periodically replicated storage tier.

Today, enterprises can exploit different storage components, configurations, and capabilities to create tiers of storage that match cost with data value, and I/O performance with data access needs, while meeting business application requirements for data availability and security.

The challenge in using multi-tier storage effectively

Fundamentally, enterprises organize their digital information as hierarchies (directories) of files. Files are usually closely associated with business purpose—documents, tables of transaction records, images, audio tracks, and other digital business objects are all conveniently represented as files, each with a business value. Files are therefore obvious objects around which to optimize storage and I/O cost and performance.

The key to deriving value from multiple tiers of storage is to place each file on the appropriate type of storage device. More critical files should be placed on higher-performing, more reliable (and therefore more expensive) devices; less critical files can be placed on less costly ones.

The problem of matching a file to the ‘right’ types of storage device is not technically challenging. Administrators can cause files to be created on the right type of storage, for example by assigning users or applications to specific devices. The challenge lies in the numbers, however. Getting millions of files placed on the right devices is far too time-consuming to do effectively without some form of automation.

Moreover, the right type of storage for a file changes over time. As a file ages, is accessed more or less frequently, grows or shrinks, or moves around within its file system logical name space, the right type of storage device changes. Manually relocating millions of files between tiers of storage devices would be a never-ending task.

Automation is a necessity for utilizing multiple tiers of storage effectively, and the more files an enterprise has, the more of a necessity it becomes. For most files, both access requirements and value to the enterprise change over time. For example, newly created transaction records are typically accessed frequently as orders are processed, confirmed, scheduled for shipment, and billed, and as customer inquiries are handled. As these records age, access tends to become less frequent, but they must nevertheless remain online for occasional inquiries, monthly and quarterly closings, mining, and other ancillary applications. In most cases, a data center’s average storage cost can be reduced substantially by moving data to lower performance, less expensive storage devices as it ages and is accessed less frequently.

Conventional multi-tier storage techniques

Enterprises typically utilize one of the two techniques listed in Table 1-3 to exploit multi-tier storage.

Table 1-3 Techniques for utilizing multi-tier storage

Technique	Description
Ad hoc relocation	It is common system management practice to create multiple file systems on storage devices of different types and move files between them to meet business needs. For example, a database application might keep current transactions in a file system on top-tier storage, and move 30-day old transactions to another file system on a second-tier device. As another example, when software or web page development completes, executable files or finished web pages would ordinarily be moved from developers' storage to production devices. A third example: as production storage devices grow dangerously close to full, large, inactive files might be relocated to larger, slower, less expensive devices.
Hierarchical storage management (HSM)	HSM software scans a file system periodically and migrates files that meet certain criteria (usually inactivity) to alternate storage devices (usually tape). HSM leaves stubs in the file system to indicate the locations of migrated files so that they can be restored to file system storage automatically when applications or users access them.

The advantages and limitations of these techniques are discussed in the sections that follow.

Advantages and limitations of ad hoc data relocation

Ad hoc policies for relocating files to optimize storage utilization have two important advantages that are listed in Table 1-4.

Table 1-4 Advantages of ad hoc data relocation

Advantage	Description
Precision	With custom-designed scripts that execute periodically, an IT organization can meet virtually any requirement for file relocation. If a requirement can be expressed as a sequence of operating system commands, it can be enforced by scripts that run automatically from authorized accounts.

Table 1-4 Advantages of ad hoc data relocation *(continued)*

Advantage	Description
In-place use	Applications and utilities access relocated files directly from wherever they reside with no hidden overheads as are incurred with HSM. Impact on applications from the processing and bandwidth overhead inherent in file relocation can be mitigated by controlling the times at which scripts run. When relocated files are used, I/O performance is determined by the devices on which they are located. Not infrequently, lack of interference from production data access outweighs the lower performance of second-tier storage devices.

But there are disadvantages to ad hoc data relocation as well. Table 1-5 lists two significant properties of ad hoc data relocation that limit its usefulness.

Table 1-5 Limitations of ad hoc data relocation

Limitation	Description
Administrative effort	While ad hoc file placement policies are flexible, each one must be created and maintained individually. A simple policy change, such as relocating transaction data after 45 rather than 30 days of inactivity, can ripple through database stored procedures and applications as well as scripts that relocate and operate on relocated data. Each time an application's data configuration changes, or the target devices for relocation change, all applications and scripts that are affected by the change must be adjusted and tested as well.
Procedural complexity	When files are relocated to alternate file systems, they are no longer accessible by applications and management processes designed to work with the original name space. Applications and operating procedures must be modified to use relocated files, and the modifications must be verified (and possibly altered) after every relocation policy change.

Thus, by employing custom-designed techniques for locating files on the right type of storage, an enterprise can effectively utilize two or more tiers of online storage devices, but at the cost of operational complexity and susceptibility to procedural errors, especially when changes in the operating environment occur.

Advantages and limitations of hierarchical storage management

Hierarchical storage management does not suffer from either of the disadvantages of ad hoc file relocation. HSM policies are standardized (move files that haven't been accessed for `x` days to location `y`, where location `y` is typically, but not necessarily, tape storage), and HSM file system scanners run on regular schedules. Stubs of relocated files remain in the file system metadata, so applications and operating procedures can typically run without modification unless they are sensitive to initial access performance, sometimes called time to first byte. Backup managers are typically HSM-aware, so that, for example, migrated files are not backed up redundantly, and so that backup of migrated files does not result in upward migration.

The three properties listed in Table 1-6 limit the general applicability of HSM, however.

Table 1-6 Limitations of HSM

Limitation	Description
Access time	When a user or application accesses a migrated file, it must be migrated back into the file system name space before the access request can be satisfied. Some HSM software attempts to mitigate the impact of reverse migration, for example by signaling completion of an application request to open a migrated file after relocating the first few blocks, and continuing to migrate data in the background. Nevertheless, there is inevitable delay and run-time overhead, particularly if migrated files must be retrieved from tape storage.
Inflexible policies	Generally, the policy options available with HSM packages are limited to migration based on complete inactivity. Policies can be configured to migrate files that have not been accessed for a designated period, but migration based on renaming, size and ownership changes, and so forth, is typically not possible.
Space management	A file system from which files have been migrated must retain adequate free space to hold migrated files that are accessed by applications. This imposes a burden on system administrators because there is no automatic advance warning that space is required for migration of one or more large files.

Thus, the two conventional techniques for utilizing multi-tier storage have both attractions and shortcomings. Ad hoc relocation policies are precise, but fragile and maintenance intensive. HSM is typically limited to inactivity-based policies, can incur significant delays when applications access relocated files, and requires management of online space to accommodate restoration of migrated files.

Hardware-based multi-tier storage facilities

Recently, storage hardware vendors have introduced facilities for relocating the contents of disk array logical units to other logical units with different properties, either on demand or on regular schedules. While these facilities are attractive because they incur little or no host I/O overhead, they are somewhat inflexible in that relocation is based on logical units (LUN) presented by the disk array. Disk arrays have no visibility of individual files or directories. To use hardware-based relocation, administrators must organize their data so that files to be relocated reside on LUNs for which relocation schedules are in effect. The administrative effort required and the susceptibility to changes in operating procedures are similar to those required for purely ad hoc multi-tier storage management.

An ideal multi-tier storage facility

It is relatively easy to acquire and configure multiple tiers of online storage. As this chapter demonstrates, however, deriving benefit from a multi-tier storage strategy requires that each data object be located on the appropriate tier of storage at all times without incurring excessive administrative cost or imposing functional or performance limitations on IT operations. An ideal facility for utilizing multiple storage tiers effectively would have the characteristics listed in Table 1-7.

Table 1-7 Characteristics of an ideal multi-tier storage facility

Characteristic	Description
Policy options	More flexible than those offered by HSM packages
Automation	Completely automatable, including scheduling of file relocation among tiers so that production application disruption is minimal
Transparency	No functional or latency effects such as time to first byte delay when applications access relocated files
Function	No limitation imposed on advanced storage device functions such as snapshots and remote replication

These are precisely the properties of the VxFS Dynamic Storage Tiering facility, a feature of the Storage Foundation that represents an evolution of the former Quality of Storage Service (QoSS) feature.

Part I of this book describes the properties and usage of the Dynamic Storage Tiering facility.

Part II describes how the Dynamic Storage Tiering facility can be employed to solve common multi-tier storage management problems in data centers.

Part I

The technology: VxFS multi-volume file systems and the Dynamic Storage Tiering facility

Using multi-tier storage

This chapter includes the following topics:

The Dynamic Storage Tiering facility

The VxFS File System enables effective exploitation of multi-tier storage through its Dynamic Storage Tiering (DST) facility. DST has two parts: support for multi-volume file systems and automatic policy-based placement of files within the storage managed by a file system.

Multi-volume file systems, as the name implies, are file systems that occupy two or more virtual storage volumes. A VxFS multi-volume file system presents a single name space, making the existence of multiple volumes transparent to users and applications. But VxFS remains aware of each volume's identity, making it possible to control the locations at which individual files are stored.

Properties of multi-volume file systems

VxFS is able to construct file systems whose logical name spaces are distributed across multiple VxVM virtual volumes. The volumes on which a file system is constructed are known as its volume set. The volumes in a volume set are configured from disks or disk array LUNs that belong to a single VxVM disk group. Volumes may be of different types (for example, striped, RAID-5, mirrored, and so forth) and may be based on different hardware technologies such as Fibre Channel disk array LUNs, SATA disk array LUNs, parallel SCSI JBOD, and so forth.

The fact that a VxFS multi-volume file system occupies multiple volumes is transparent to applications. All files in the file system are part of the same name space and are accessed and manipulated as though they all occupied a single volume.

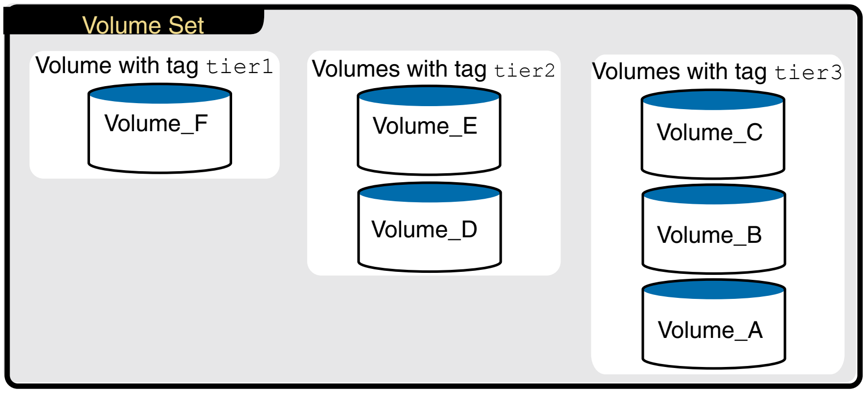
Basing storage tiers on virtual volumes offers an important advantage over hardware-based multi-tier storage approaches. Unlike hardware-based solutions whose tiers are based on LUNs presented by disk arrays, Storage Foundation virtual volumes can be of any required capacity and configuration, even spanning multiple disk arrays if required for I/O performance or failure tolerance reasons.

Using placement classes to manage file locations

Administrators of multi-volume VxFS file systems can control the locations of files within volume sets by defining file placement policies that control both initial file location and the circumstances under which existing files are relocated. A VxFS file placement policy consists of rules that restrict the locations of files to administrator-defined subsets of the volumes in a file system's volume set. These subsets are called placement classes. A placement class is typically identified with a storage tier. Policy rules cause files to be created and extended within specified placement classes, and to be relocated to other placement classes when they meet certain naming, activity, access rate, and size-related qualifications.

As an example, Figure 2-1 represents a VxFS multi-volume file system whose volume set consists of three placement classes called `tier1`, `tier2`, and `tier3`. Such a volume set might be suitable for a file system containing a few critical files (`tier1`), a larger number of files of average importance (`tier2`), and a still larger number of inactive files (`tier3`).

Figure 2-1 Example volume set for a multi-volume file system



An administrator assigns VxVM volumes to placement classes by associating character strings called volume tags with them. In Figure 2-1 `Volume_F` is tagged `tier1`, `Volume_D` and `Volume_E` are tagged `tier2`, and `Volume_A`, `Volume_B`, and `Volume_C` are tagged `tier3`. For file placement purposes, VxFS treats all of the volumes in a single placement class as equivalent, and balances space allocation approximately equally across them.

To the VxFS file system, a volume tag is simply a character string used to classify a volume. Any storage tier may be tagged with any convenient name. For example, some enterprises name their storage tiers after precious metals—`gold` (uppermost tier), `silver` (middle tier), and `bronze` (lowest tier).

VxFS imposes no capacity, performance, availability, or other constraints on placement classes. Any volume may be added to any placement class by assigning to it the tag that identifies the class, no matter what its type or the types of other volumes in the class. Practically speaking, however, it is good practice to place only volumes with identical, or at least very similar, I/O performance and availability characteristics in a single placement class; in other words, to identify a placement class with a physical storage tier.

File placement policies

VxFS places files among the volumes of a file system's volume set in accordance with the file system's active file placement policy. A file placement policy consists of rules that govern the initial location and subsequent relocation of designated sets of files. A rule may designate the files to which it applies by name, by directory, by ownership, or by combinations of the three.

Policy rules specify where files should be placed in terms of placement classes rather than specific volumes. This makes it unnecessary to change a file system's active placement policy when volumes are added to or removed from its volume set. Moreover, because the volume tags that define placement classes need not be unique, one placement policy can be used for any number of file systems with similar requirements and storage complements.

Policy rules specify both initial allocation destinations and relocation destinations as priority-ordered lists of placement classes. Files are allocated in the first placement class in the list if free space permits, in the second class if no free space is available in the first, and so forth.

File placement policy enforcement

File relocation is performed when a policy is enforced, either on-demand or periodically, and is similar to initial allocation. Files are relocated to the first placement class listed in the rule that selects them if space is available, to the second class if no space is available in the first, and so forth.

File relocation may be unconditional, or it may be based on qualifications such as time since most recent access or modification, intensity of access by applications (I/O temperature), and file size. A file system's policy for allocating and relocating files is expressed in a set of internal data structures called its active file placement policy.

Administrators write file placement policies in the XML language, according to a Document Type Description (DTD) supplied with VxFS. The Storage Foundation graphical management console includes wizards that create four popular types of policies in accordance with user-supplied parameters.

File placement policies are not inherently bound to specific file systems. An administrator assigns a policy to a file system, making it the file system's active policy. A file system may have only one active policy at a time. When assigned to a file system, a file placement policy allocates and relocates files among the placement classes that are named in the policy and represented by tags assigned to the volumes.

Chapter 3

The VxFS file placement policy grammar

This chapter includes the following topics:

VxFS file placement principles

VxFS allocates and relocates files among the volumes of a multi-volume file system based on properties expressed in file system metadata. Initial allocation may be based on any or all of file name, logical location within the name space (directory), and file ownership. Relocation may be based on the same criteria, and in addition, on qualifications such as time since most recent access or modification, intensity with which they are being accessed by applications (I/O temperature), and file size. A file system's policy for allocating and relocating files is expressed in a set of internal data structures called its active file placement policy.

VxFS file placement policies

A VxFS file system may have one active file placement policy. A file system's active policy defines the desired placement of files on the volumes that comprise its volume set. A file placement policy specifies the placement classes of volumes on which files should be placed when they are created and conditions under which the files should be deleted or relocated to other placement classes. Using an XML editor, a text editor, or a VEA graphical console wizard, administrators create file placement policy documents, which are text files in the XML markup language. An administrator uses a console command or graphical interface to assign a file placement policy to a file system, making it the file system's active policy, and superseding the previous active policy if there was one. For example, illustrates the use of the `fsppadm assign` command to assign an active file placement policy.

Dialog 3-1 Assigning a file system's active file placement policy

```
# /opt/VRTS/bin/fsppadm assign /vsb /tmp/policy1.xml
```

The command in Dialog 3-1 assigns the placement policy in `/tmp/policy1.xml` as the active policy for the file system mounted at `/vsb`.

Fragment 3-1 shows the general structure of a VxFS file placement policy document.

Fragment 3-1 General form of a VxFS file placement policy

```
<?xml version="1.0"?>
<!DOCTYPE FILE_PLACEMENT_POLICY SYSTEM "/opt/VRTSfspro/config/placement_policy.dtd">
<FILE_PLACEMENT_POLICY Version="5.0">
  <RULE Name=rule_name>
    <SELECT>...selection criterion...</SELECT>
    <SELECT>...repeated as necessary...</SELECT>
    <CREATE><ON>...creation placement class(es)... </ON></CREATE>
    <DELETE><FROM>...source placement class(es)...</FROM>
           <WHEN>...deletion qualifiers...</WHEN>
    </DELETE>
    <DELETE>...repeated as necessary...</DELETE>
    <RELOCATE><FROM>...source placement class(es)...</FROM>
            <TO>...destination placement class(es)...</TO>
            <WHEN>...relocation qualifiers...</WHEN>
    </RELOCATE>
    <RELOCATE>...repeated as necessary...</RELOCATE>
  </RULE>
  <RULE Name=rule_name>...repeated as necessary...</RULE>
</FILE_PLACEMENT_POLICY>
```

A VxFS file placement policy contains rules according to which VxFS creates, relocates, and deletes selected sets of files, but does not refer to specific file systems or specific volumes. A VxFS file system may have at most one active file placement policy at any time, although the policy may contain an unlimited number of rules that apply to different sets of files. A file system may have no active file placement policy assigned to it, in which case VxFS controls space allocation for new files, and no file relocation occurs.

File placement policies and Storage Foundation Management Server

File placement policies can be administered either directly on the system to which they apply, or centrally for an entire data center or enterprise using the Storage Foundation Management Server (SFMS). SFMS is a client-server suite for consolidated management of storage on multiple platforms. For systems with SFMS client software installed, file placement policies can be stored in the Management Server's database along with lists of file systems in which each policy is active. When a policy is updated, SFMS can assign the updated policy to file systems whose active policies are based on it. By default, however, SFMS does not update active policies that have been created or modified locally at their file systems' hosts. If a SFMS administrator forces assignment of a policy from its database to a file system, that file system's prior active policy is overwritten and any local changes that had been made to it are lost.

File placement policy rules

As Fragment 3-1 suggests, a VxFS file placement policy (lines `[3]-[19]`) essentially consists of one or more rules (lines `[4]-[17]`). Each rule consists of statements that apply to a part of the file system's name space. The files to which a rule applies are specified in one or more `SELECT` statements within the rule (lines `[5]-[6]`). Files specified in a `SELECT` statement are said to be selected by the rule. A `SELECT` statement specifies files according to one or more of four properties: their names or naming patterns, the directories in which they reside, and the user or group names of their owners.

A file may be selected by more than one rule. For example, if one rule selects files in directory `/dir`, and another selects files owned by `user1`, a file in `/dir` that is owned by `user1` is selected by both rules. In this situation, only the rule that appears first in the policy is applied.

It is possible to define policies that do not encompass an entire file system name space. When an application creates a file that is not selected by any rule in its file system's active policy, VxFS chooses the file's location. To maintain full control over file placement, administrators should include a 'catchall' rule as the final rule in each policy, with a `SELECT` statement containing a `<PATTERN>*</PATTERN> clause`, that selects all files in the file system. Such a rule selects all files that have not already been selected by some rule appearing earlier in the policy.

The SELECT statement

A `SELECT` statement in a policy rule specifies the file or files to which the rule applies. A policy rule must contain one or more `SELECT` statements. Fragment 3-2 illustrates the general form of the `SELECT` statement.

Fragment 3-2 General form of the SELECT statement

```
<SELECT>
  (optional) <DIRECTORY Flags=directory_flag_value>...value...</DIRECTORY>
  ...(optional) additional directory specifications...
  optional) <PATTERN>...value...</PATTERN>
  ...(optional) additional pattern specifications...
  optional) <USER>...value...</USER>
  ...(optional) additional owner's username specifications...
  (optional) <GROUP>...value...</GROUP>
  ...(optional) additional owner's groupname specifications...
</SELECT>
```

As Fragment 3-2 suggests, a `SELECT` statement may designate files by any or all of the four selection criteria listed in Table 3-1.

Table 3-1 SELECT statement file selection criteria

Criterion	Description
`<DIRECTORY>`	Specifies a full path name relative to the file system mount point. The value of the required `Flags` attribute of the `<DIRECTORY>` tag must either be `nonrecursive,` meaning that only files in the specified directory are selected, or `recursive`, meaning that files in all subdirectories of the specified directory are selected.
`<PATTERN>`	Specifies either a file name or a pattern that includes a single wildcard character (`*`)representing don't care characters in a name. The first `*` character that appears in the pattern is interpreted as the wildcard; others are interpreted as elements of the file name. For example: ■ The pattern `abc*` denotes all files whose names begin with `abc`, whether or not the names have extensions ■ The pattern `abc.*` denotes all files whose names are exactly `abc` followed by a period and any extension ■ The pattern `*abc` denotes all files whose names end in `abc`, whether it is all or part of an extension or not ■ The pattern `*.abc` denotes files of any name whose name extension (following the last period) is exactly `abc` ■ The pattern `ab*c` denotes all files whose names start with `ab` and end with `c` ■ The pattern `abc*.*` denotes all files whose names start with `abc` and whose extensions following the last period are exactly the character `*`
`<USER>`	Owning username (user number may not be specified).

Table 3-1 SELECT statement file selection criteria *(continued)*

Criterion	Description
`<GROUP>`	Owning groupname (group number may not be specified).

One or more instances of any or all of the file selection criteria listed in Table 3-1 may be specified within a single `SELECT` statement. If two or more selection criteria of different types are specified in a single statement, a file must satisfy one criterion of each type in order to be selected. A file that meets one of each of the types of criteria specified in a `SELECT` statement is said to be selected by the rule.

SELECT statement examples

For example, Fragment 3-3 shows a single `SELECT` statement that specifies both `<USER>` and `<DIRECTORY>` file selection criteria.

Fragment 3-3 SELECT statement containing multiple selection criteria

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">ora/db</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">crash/dump</DIRECTORY>
  <USER>user1</USER>
  <USER>user2</USER>
</SELECT>
```

Only files that both reside in one of the specified directories and are owned by one of the specified users are selected by the rule of which this statement is a part. For example, a file in the `ora/db` directory whose owner is `user1` is selected, but a file in `crash/dump` that is owned by `user3` is not.

A rule may include multiple `SELECT` statements. If a file satisfies the selection criteria of one of the `SELECT` statements, it is selected by the rule. For example, if one `SELECT` statement specifies file owners (`<USER>`) and another specifies directories (`<DIRECTORY>`), as in Fragment 3-4, files owned by specified users, no matter which directories they reside in, as well as all files in specified directories, no matter which users own them, are selected by the rule containing the `SELECT` statements.

Fragment 3-4 Two SELECT statements in a file placement policy rule

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">ora/db</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">crash/dump</DIRECTORY>
</SELECT>
<SELECT>
  <USER>user1</USER>
```

Fragment 3-4 Two SELECT statements in a file placement policy rule *(continued)*

```
[7]         <USER>user2</USER>
[8]       </SELECT>
```

When VxFS creates new files, it applies the rules in the file system's active policy in the order of their occurrence. The first rule by which a file is selected determines the file's placement; no later rules apply. Similarly, when VxFS relocates files, it evaluates the file system's active policy rules in order on behalf of each file, stopping when it reaches the first rule that selects the file, even if that rule results in no action. For example, if a rule specifies that `.dat` files should be relocated to `tier2` volumes if they are not accessed for 30 days, and a later rule indicates that `.dat` files should be relocated to `tier3` volumes if they are larger than 10 megabytes, a 20 megabyte `.dat` file that was accessed 10 days ago is not relocated, because the earlier rule selects it, so the later rule is never evaluated on its behalf.

A policy rule's action statements apply to all files selected by any of its `SELECT` statements. If there is no rule in a file system's active policy that selects an already-existing file, the Dynamic Storage Tiering facility does not relocate or delete the file. If an application creates a file that is not selected by any rule in the file system's active policy, VxFS chooses the file's initial location. If this behavior is inappropriate, the last rule in the file system's policy should select all files by using the `<PATTERN>*</PATTERN>` clause as the only criterion in its `SELECT` statement, and should include a `CREATE` action statement naming the desired placement class(es) for files not selected by rules appearing earlier in the policy.

File placement policy enforcement

VxFS follows a file system's active policy when applications create new files, by allocating space for them in placement classes designated by the applicable policy rule. To enforce the file relocation and deletion aspects of the active placement policy administrators can run the `fsppadm enforce` console command at any time. The `fsppadm enforce` command is typically scheduled to run automatically from an authorized user account at regular intervals, for example at off-peak times for production applications that use the target file system. The command may also be run interactively by authorized users whenever file system conditions warrant it. In this book, phrases such as enforce the active policy denote running the `fsppadm enforce` command to enforce file relocation and deletion policy rules.

The action statements in file placement policy rules

A file placement policy rule governs the files specified in one of its `SELECT` statements throughout the parts of their lifecycles that are spent within the file system. (Most files of importance spend the latter parts of their lifecycles as archives outside the file systems in which they were created and processed. Dynamic Storage Tiering does not affect that part of the lifecycle.)

A rule designates the placement class(es) of volumes on which selected files may be created, circumstances under which they qualify for automatic deletion, and circumstances under which they qualify for relocation to other placement classes, using the `CREATE` (line `[7]` in Fragment 3-1), `DELETE` (lines `[8]-[11]` in Fragment 3-1), and `RELOCATE` (lines `[12]-[16]` in Fragment 3-1) statements respectively. A file placement policy may contain zero or one `CREATE` statements, and any number (including zero) of `DELETE` and `RELOCATE` statements.

The actions performed by each of the three placement policy action statements are regulated by clauses whose functions are summarized in Table 3-2.

Table 3-2 Policy rule action statement clauses

Clause	Description
`CREATE` statement	Must contain one `<ON>` clause, which lists one or more placement classes in priority order. VxFS creates files selected by the rule in the placement classes named in this clause. If VxFS cannot allocate space for a new file in one of the designated placement classes, file creation fails unless the `Flags=any` attribute is specified in the `<ON>` clause. If a rule does not contain a `CREATE` statement, its only effect is to relocate or delete existing files when the policy is enforced.

Table 3-2 Policy rule action statement clauses *(continued)*

Clause	Description
`DELETE` statement	May contain a `<FROM>` clause listing the placement classes of volumes containing files selected by the rule. If no `<FROM>` clause is present, VxFS deletes qualifying files located anywhere in the file system's volume set when the policy is enforced. May contain a `<WHEN>` clause defining circumstances under which files qualify for deletion. If no `<WHEN>` clause is present, all files that are selected by the rule and that reside in placement classes specified in the `<FROM>` clause are deleted when the policy is enforced. If a `DELETE` statement contains neither a `<FROM>` clause nor a `<WHEN>` clause, all files to which the containing rule applies are deleted. A rule may contain any number of `DELETE` statements, including zero. During policy enforcement, VxFS only acts upon the first `DELETE` or `RELOCATE` statement that applies to a file.
`RELOCATE` statement	May contain a `<FROM>` clause listing the placement classes of volumes containing files selected by the rule. If no `<FROM>` clause is present, VxFS relocates qualifying files from anywhere in the file system's volume set. Must contain one `<TO>` clause listing in priority order the placement classes of volumes to which VxFS relocates qualifying files when the policy is enforced. May contain a `<WHEN>` clause defining circumstances under which files qualify for relocation. If no `<WHEN>` clause is present, VxFS relocates all files that are selected by the rule and that reside on volumes specified in the `<FROM>` clause. A rule may contain any number of `RELOCATE` statements, including zero, but during policy enforcement, VxFS only acts upon the first `RELOCATE` or `DELETE` statement that applies to each file.

The sections that follow describe the three policy rule action statements in more detail.

The CREATE statement

A file placement policy rule may contain a `CREATE` statement. If present, the `CREATE` statement specifies one or more placement classes in which VxFS creates files selected by the rule. Only placement classes may be specified in a CREATE statement, not individual volume names. This property makes it possible to add volumes to a file system's volume set without changing its active file placement policy. Moreover, the property makes it possible for Storage Foundation Management Server to apply a single policy to multiple file systems on different hosts, promoting storage management policy standardization throughout large data centers.

A rule may contain at most one CREATE statement. VxFS places new files selected by a rule that does not contain a CREATE statement according to internal algorithms.

Fragment 3-5 shows the general form of the CREATE statement.

Fragment 3-5 General form of the CREATE statement

```
[1]      <CREATE>
[2]        <ON Flags=...flag_value...>
[3]          <DESTINATION>
[4]            <CLASS>...placement_class_name...</CLASS>
[5]            <BALANCE_SIZE Units="units_specifier">...chunk_size...</BALANCE_SIZE>
[6]          </DESTINATION>
[7]          <DESTINATION>...additional placement class specifications...</DESTINATION>
[8]        </ON>
[9]      </CREATE>
```

The <ON> clause

A CREATE statement must include a single <ON> clause (line [2] in Fragment 3-5). The <ON> clause contains one or more <DESTINATION> elements that, together with their <CLASS> sub-elements, specify placement classes for initial file allocation in order of decreasing preference. VxFS allocates space for new files in the first listed placement class that has sufficient free space available.

If space cannot be allocated in any placement class in the list, file creation fails with an ENOSPC indication, even if adequate space is available elsewhere in the file system's volume set. Allocation failure can be averted by specifying the Flags=any attribute in the <ON> clause. If a CREATE statement specifies <ON Flags=any>, VxFS first attempts to allocate space for new files on one of the specified placement classes. Failing that, it allocates space elsewhere in the file system's volume set, so file allocation does not fail unless there is no available space anywhere in the file system's volume set.

The Flags=any attribute differs from the catchall rule described earlier in that it applies only to files selected by the rule in which it appears, whereas the file selection specification of the catchall rule includes all files not selected by earlier rules.

The <BALANCE_SIZE> sub-element

In addition to the `<CLASS>` sub-element, a `<DESTINATION>` element may contain a `<BALANCE_SIZE>` sub-element. When an `<ON>` clause contains a `<BALANCE_SIZE>` sub-element, VxFS allocates extents of the indicated size (rounded up to the nearest multiple of the file system block size) for files selected by the rule, distributing its allocations approximately uniformly across the volumes in the placement class. Distributing a large file across multiple volumes may provide performance benefits for both transactional and streaming access.

For example, a balance size of one megabyte for a `<DESTINATION>` placement class containing three volumes causes VxFS to allocate space for new files in one-megabyte chunks, randomly distributed across the three volumes in the placement class.

VxFS supports one balance size per placement class. The first `<BALANCE_SIZE>` sub-element in a policy becomes the balance size for the placement class to which it applies. Using the `Units` attribute of the `<BALANCE_SIZE>` sub-element, balance size may be specified in bytes (`Units=bytes`), kilobytes (`Units=KB`), megabytes (`Units=MB`), or gigabytes (`Units=GB`).

Balancing space allocation across the volumes in a placement class has an effect similar to virtual volume block address striping. Distributing a file's data storage across multiple volumes makes it possible to use multiple I/O resources concurrently, potentially improving I/O performance for both transactional and streaming applications. Balanced allocation is particularly useful with database management systems and other applications that allocate storage in the form of small numbers of large files. Fragment 3-6 illustrates a sample file placement policy `CREATE` statement that includes the `<BALANCE_SIZE>` sub-element in one of its two placement class specifications.

Fragment 3-6 Sample CREATE statement

```
<CREATE><ON>
  <DESTINATION><CLASS>tier1</CLASS></DESTINATION>
  <DESTINATION>
    <CLASS>tier2</CLASS>
    <BALANCE_SIZE Units="MB">1</BALANCE_SIZE>
  </DESTINATION>
</ON></CREATE>
```

The `CREATE` statement in Fragment 3-6 specifies that files selected by the rule be created on `tier1` volumes if space is available, and on `tier2` volumes if not. Because the `<ON>` clause does not include `Flags=any`, file creation fails if no space is available on any `tier1` or `tier2` volume, even if space is available elsewhere in the file system's volume set.

The balance size of one megabyte applies only to allocations on tier2 volumes. VxFS does not balance files placed on tier1 volumes. For files placed on tier2, VxFS allocates successive one-megabyte extents randomly among the tier2 volumes.

The RELOCATE statement

The DELETE and RELOCATE action statements specify actions that VxFS takes on qualifying files during policy enforcement, as well as the circumstances that qualify files to be acted upon. The two statements are quite similar in form and function; indeed, DELETE can be thought of as RELOCATE with the null device as a destination.

During policy enforcement, VxFS scans file systems in path name order, stopping when it calculates that already-scheduled relocations would result in destination volumes being fully occupied. For each file, VxFS identifies the first applicable rule in the active policy as determined by the rules' SELECT statements. If the file resides in a placement class specified in the <FROM> clause of one of the rule's RELOCATE statements, and if it meets the qualifications for relocation specified in the statement's <WHEN> clause, VxFS relocates it to the first placement class listed in the <TO> clause that has space available for it. Only placement classes may be specified in <FROM> and <TO> clauses, not individual volume names.

Fragment 3-7 shows the general form of the RELOCATE statement. Table 3-3 lists and describes the RELOCATE statement's action clauses.

Fragment 3-7 General form of the RELOCATE statement

```
<RELOCATE>
  <FROM>
    <SOURCE><CLASS>...placement_class_name...</CLASS></SOURCE>
    <SOURCE>...additional placement class specifications...</SOURCE>
  </FROM>
  <TO>
    <DESTINATION>
      <CLASS>...placement_class_name...</CLASS>
      <BALANCE_SIZE Units="units_specifier">...chunk_size...</BALANCE_SIZE>
    </DESTINATION>
    <DESTINATION>...additional placement class specifications...</DESTINATION>
  </TO>
  <WHEN>...relocation conditions...</WHEN>
</RELOCATE>
```

Table 3-3 Clauses in the RELOCATE statement

Clause	Description
`<FROM>`clause (lines `[2]-[5]`)	The optional `<FROM>` clause specifies the placement class or classes from which files selected by the rule should be relocated. If a file selected by the rule is located in any specified placement class, VxFS relocates it, provided that it meets the qualifications specified in the `<WHEN>` clause.
`<TO>` clause (lines `[6]-[12]`)	The required `<TO>` clause specifies a priority ordered list of one or more placement classes to which VxFS relocates qualifying files. VxFS relocates files to the first specified placement class if space permits, to the second if not, and so forth.
`<WHEN>` clause (line `[13]`)	The optional `<WHEN>` clause specifies additional qualifications for relocation. Files may be relocated when they have been inactive or unmodified for specified periods, when they reach a specified size, or when their I/O temperatures or access temperatures (levels of recent application I/O activity) reach a specified level. If a `RELOCATE` statement does not contain a `<WHEN>` clause, files selected by the rule are relocated unconditionally.

If a `RELOCATE` statement contains a `<FROM>` clause, VxFS relocates qualifying files only if they reside in placement classes specified in the `<FROM>` clause. If no `<FROM>` clause is present, VxFS relocates qualifying files regardless of their locations.

The required `<TO>` clause of the `RELOCATE` statement contains a priority-ordered list of `<DESTINATION>` elements specifying placement classes to which VxFS relocates qualifying files selected by the rule. VxFS relocates files to the first specified placement class if space is available, to the second specified class if no space is available in the first, and so forth.

The <BALANCE_SIZE> sub-element

A `<DESTINATION>` element in a `<TO>` clause may contain an optional `<BALANCE_SIZE>` sub-element. The `<BALANCE_SIZE>` sub-element causes VxFS to distribute space for relocated files approximately evenly across destination placement class volumes in extents of the indicated size. VxFS supports one balance size per placement class. The first `<BALANCE_SIZE>` sub-element in a policy becomes the balance size for the placement class to which it applies. Using the `Units` attribute in its XML tag, the `<BALANCE_SIZE>` value may be specified in bytes (`Units=bytes`), kilobytes (`Units=KB`), megabytes (`Units=MB`), or gigabytes (`Units=GB`).

Specifying a `<BALANCE_SIZE>`has an effect similar to virtual volume block striping. The capability is particularly useful with databases and other applications that store data in the form of small numbers of large files. Striping files allows concurrent use of multiple physical I/O resources, potentially improving performance for both transactional and streaming applications.

The <WHEN> clause in a RELOCATE statement

An optional `<WHEN>` clause may be included in a `RELOCATE` statement. The `<WHEN>` clause restricts relocation to selected files that meet certain qualifications. Fragment 3-8 shows the general form of the `WHEN` clause in a `RELOCATE` statement.

Fragment 3-8 General form of the WHEN clause in a RELOCATE statement

```
<WHEN>
  <ACCAGE Units="...units_value...">
    <MIN Flags="...comparison_operator...">...min_access_age...</MIN>
    <MAX Flags="...comparison_operator...">...max_access_age...</MAX>
  </ACCAGE>
  <MODAGE Units="...units_value...">
    <MIN Flags="...comparison_operator...">...min_modification_age...</MIN>
    <MAX Flags="...comparison_operator...">...max_modification_age...</MAX>
  </MODAGE>
  <SIZE " Units="...units_value...">
    <MIN Flags="...comparison_operator...">...min_size...</MIN>
    <MAX Flags="...comparison_operator...">...max_size...</MAX>
  </SIZE>
  <IOTEMP Type="...read_write_preference...">
    <MIN Flags="...comparison_operator...">...min_I/O_temp...</MIN>
    <MAX Flags="...comparison_operator...">...max_I/O_temp...</MAX>
    <PERIOD>...days_of_interest...</PERIOD>
  </IOTEMP>
  <ACCESSTEMP Type="...read_write_preference...">
    <MIN Flags="...comparison_operator...">...min_access_temp...</MIN>
    <MAX Flags="...comparison_operator...">...max_access_temp...</MAX>
    <PERIOD>...days_of_interest...</PERIOD>
  </ACCESSTEMP>
</WHEN>
```

Relocation qualifiers

If a `RELOCATE` statement contains a `<WHEN>` clause, files selected by the rule qualify for relocation only if they satisfy all of the qualifications that are specified in the `<WHEN>` clause. Table 3-4 lists the five qualifications that may be specified in a `<WHEN>` clause.

Table 3-4 Relocation qualifiers in the WHEN clause of the RELOCATE statement

Qualifier	Description
`<ACCAGE>`	Qualifies files that have not been accessed for a specified period immediately prior to the time at which the policy is enforced.
`<MODAGE>`	Qualifies files that have not been modified for a specified period immediately prior to the time at which the policy is enforced.
`<SIZE>`	Qualifies files that exceed or drop below a specified size or fall within a specified size range.
`<IOTEMP>`	Qualifies files that exceed or drop below a specified average I/O temperature, or fall within a specified I/O temperature range. A file's I/O temperature, described in The <IOTEMP> (I/O temperature) relocation qualifier , is a measure of the average I/O activity against it, measured in bytes transferred, during a specified period immediately prior to policy enforcement.
`<ACCESSTEMP>`	Qualifies files that exceed or drop below a specified average access temperature, or fall within a specified access temperature range. A file's access temperature is similar to its I/O temperature, except that it is computed using the number of I/O requests to the file rather than the number of bytes transferred.

The sections that follow discuss these five qualifiers in more detail.

The access age (<ACCAGE>) relocation qualifier

A file's access age is the interval between the policy enforcement time and the last application access to the file. VxFS computes access age by subtracting a file's time of most recent access (its POSIX `atime`) from the time at which the policy is enforced. The `<MIN>` and `<MAX>` sub-elements denote the minimum and maximum access age thresholds for relocation. At least one of `<MIN>` and `<MAX>` must be specified. Files selected by the rule qualify for relocation if their access ages are greater than the `<MIN>` value (if one is specified) and less than the `<MAX>` value (if one is specified).

If specified, the `<MIN>` and `<MAX>` sub-elements require `Flags` attributes to define their operation. For `<MIN>`, any of `Flags=gt`, `Flags=eq`, and `Flags=gteq` may be specified, with the obvious meanings. For `<MAX>`, either `Flags=lt` or `Flags=lteq` may be specified. Using the `Units` attribute, access age thresholds may be specified in seconds (`Units=seconds`), minutes (`Units=minutes`), hours (`Units=hours`), or days (`Units=days`), where a day is a 24 hour period immediately preceding the start of policy enforcement.

The <MIN> sub-element can be used to relocate inactive files to lower tiers in a storage hierarchy. Conversely, the <MAX> sub-element can be used to relocate recently active files to upper storage tiers.

The modification age <MODAGE> relocation qualifier

The modification age relocation qualifier (<MODAGE>) is identical to the access age one, except that files' POSIX `mtime` values are used in qualifying computations. The same <MIN> and <MAX> sub-elements, and `Flags` and `Units` attributes are used. Administrators might specify the <MODAGE> qualifier to relocate recently modified files to higher storage tiers in anticipation that they would be accessed again in the near future.

The <SIZE> relocation qualifier

The <SIZE> relocation qualifier enables files selected by the rule to be relocated if they are larger than the <MIN> or smaller than the <MAX> threshold when the policy is enforced. Specifying both <MIN> and <MAX> causes VxFS to relocate files selected by the rule whose sizes lie between the two. Using the `Units` attribute, threshold file sizes may be specified in bytes (`Units=bytes`), kilobytes (`Units=KB`), megabytes (`Units=MB`), or gigabytes (`Units=GB`).

The <IOTEMP> (I/O temperature) relocation qualifier

The <IOTEMP> relocation qualifier enables files selected by the rule to be relocated if their average I/O temperatures rise above or drop below specified values over a specified period immediately prior to policy enforcement. Calculation of a file's I/O temperature is discussed in "Calculating a file's I/O temperature and access temperature " on page 47., but generally speaking, a file's I/O temperature is a smoothed measure of recent read, write, or total data transfer activity against it. VxFS computes a file's average I/O temperature by calculating the average number of bytes per day transferred to or from it (or both) during the specified period, and dividing that number by the file's size. Higher I/O temperatures indicate higher levels of sustained application activity; lower temperatures indicate less activity.

As with other file relocation qualifiers, a lower <IOTEMP> threshold may be specified by using the <MIN> sub-element, an upper threshold with the <MAX> sub-element, or a range by using both. I/O temperature is dimensionless so the <MIN> and <MAX> sub-elements have no units attribute. The required <PERIOD> sub-element specifies the number of days over which VxFS computes average I/O temperature.

Using the `Type` attribute of the <IOTEMP> element, an administrator can constrain I/O temperature calculations to include only read activity (`Type=nrbytes`), write activity (`Type=nwbytes`), or both (`Type=nrwbytes`) as appropriate for the application.

Average daily I/O temperature is a smoothed measure of I/O activity compared to access or modification age. Whereas a single access to a file resets its `atime` to the time of the access, a file's I/O temperature decreases gradually as time passes without access, and increases gradually if it is accessed more often.

Figure 3-1 illustrates the gradual decrease in I/O temperature.

Figure 3-1 Contributors to I/O temperature calculation

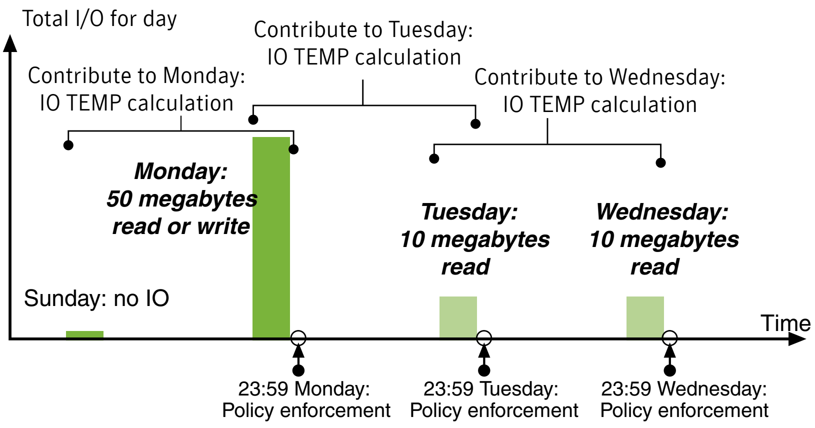


Figure 3-1 represents a ten megabyte file that is not accessed on Sunday, but is read or written completely five times on Monday. The file system's policy is enforced at 23:59 on Monday, when the file's two-day average I/O temperature is 2.5 and its access age in days is zero. If the entire file is read once on Tuesday, its access age in days at 23:59 on Tuesday is zero, but its two-day average I/O temperature is 3.0. If it is read once more on Wednesday, its access age at 23:59 on Wednesday is still zero, but its two-day I/O temperature drops to 1.0 because the influence of Monday's heavy I/O has disappeared from the calculation.

If the goal is to keep files in place as long as they are being accessed at all, then access age is the more appropriate relocation qualifier. If, however, the goal is to relocate files as I/O decreases rather than ceases entirely, then I/O temperature is more appropriate.

Upward relocation is similar. If files that have been relocated to lower-tier volumes due to infrequent access experience intensified application activity, it may be appropriate to relocate them to upper-tier devices. A policy rule that relocates based on access age, and that has a `<MAX>` value comparable to the interval between successive scheduled policy enforcements, relocates files that have been accessed even once during the <ACCAGE> interval. By contrast, a policy that uses I/O temperature with a <MIN> value only relocates files that have experienced sustained activity over the period of interest.

The <ACCESSTEMP> (access temperature) relocation qualifier

The <ACCESSTEMP> relocation qualifier enables files selected by the rule to be relocated if their average access temperatures rise above or drop below specified values over a specified period immediately prior to policy enforcement. A file's access temperature is the average number of recent read and write requests against it. As with I/O temperature, higher access temperatures indicate higher levels of sustained application activity; lower temperatures indicate lower levels.

A lower <ACCESSTEMP> threshold may be specified by using the <MIN> sub-element, an upper threshold with the <MAX> sub-element, or a range by using both. Average access temperature is always expressed in accesses per day, so the <MIN> and <MAX> sub-elements have no units attribute. The number of 24-hour periods immediately prior to policy enforcement over which access temperature is computed is specified by the required <PERIOD> sub-element.

Using the `Type` attribute of the <ACCESSTEMP> element, access temperature calculations can be constrained to include only read requests (`Type=nreads`), write requests (`Type=nwrites`), or both (`Type=nrws`) as appropriate for the application.

Average daily access temperature is a smoothed measure of I/O activity compared to access or modification age. Whereas a single access to a file resets its `atime` to the current time, a file's access temperature decreases gradually as time passes without activity, and increases gradually as activity increases. For example, if there are no read requests to a file on Sunday and 5,000 requests on Monday, and the file system policy is enforced just prior to midnight on Monday, the file's two-day average access temperature is 2,500 and its access age in days is zero. If there are 1,000 additional read requests to it on Tuesday, its access age in days just prior to midnight on Tuesday is zero, but its two-day average access temperature is 3,000. If there are 1,000 additional read requests on Wednesday, the file's access age in days just prior to midnight on Wednesday is still zero, but its two-day access temperature drops to 1,000 because the influence of Monday's I/O disappears from the calculation.

Unlike I/O temperature, access temperature is independent of file size and amount of data transfer. It is generally the more appropriate relocation qualifier for files that are accessed with many small I/O requests rather than complete data transfers.

The <PERIOD> sub-element

The value of the `<PERIOD>` sub-element is interpreted as a number of consecutive 24-hour periods immediately prior to policy enforcement. VxFS computes files' average daily I/O temperatures and access temperatures over the period between the time of policy enforcement and the number of 24-hour periods in the past specified in the `<PERIOD>` sub-element. For example, if the policy is enforced at 14:00 on Wednesday with a `<PERIOD>` value of 2, VxFS uses file I/O activity between 14:00 on Monday and 14:00 on Wednesday to compute average daily temperatures.

Although a different `<PERIOD>` value can be specified in every `<IOTEMP>` clause, VxFS currently uses only the largest value in its calculations.

RELOCATE statement example: unconditional relocation

Fragment 3-9 illustrates the simplest form of the `RELOCATE` policy rule statement—unconditional relocation.

Fragment 3-9 Unconditional RELOCATE statement

```
<RELOCATE>
  <FROM><SOURCE><CLASS>tier1</CLASS></SOURCE></FROM>
  <TO><DESTINATION><CLASS>tier2</CLASS></DESTINATION></TO>
</RELOCATE>
```

VxFS relocates files selected by this rule that reside in placement class `tier1` to placement class `tier2` as long as space permits. This type of rule might be used with applications that create and access new files but seldom access existing files once they have been processed. A `CREATE` statement would specify creation on `tier1` volumes (presumably high performance or high availability, or both). Each enforcement of the policy would relocate files created since the previous enforcement to `tier2` volumes.

RELOCATE statement example: access age-based relocation

Fragment 3-10 illustrates a more elaborate form of the `RELOCATE` statement—one that uses access age as the qualifier for relocating files from `tier1` volumes to `tier2` volumes.

Fragment 3-10 Inactivity-based downward RELOCATE statement

```
<RELOCATE>
  <FROM><SOURCE><CLASS>tier1</CLASS></SOURCE></FROM>
  <TO><DESTINATION><CLASS>tier2</CLASS></DESTINATION></TO>
  <WHEN><ACCAGE Units="days"><MIN Flags="gt">30</MIN></ACCAGE></WHEN>
</RELOCATE>
```

This statement causes VxFS to relocate files selected by the rule from `tier1` volumes to `tier2` volumes if they have not been accessed for more than 30 days. A rule containing such a statement as this would have the effect of maintaining free space on `tier1` volumes by relocating files that are inactive for more than 30 days to `tier2` volumes.

RELOCATE statement example: I/O temperature-based relocation

Fragment 3-11 illustrates a companion rule that relocates files from `tier2` volumes to `tier1` volumes based on their I/O temperatures.

Fragment 3-11 Upward RELOCATE statement

```
<RELOCATE>
  <FROM><SOURCE><CLASS>tier2</CLASS></SOURCE></FROM>
  <TO><DESTINATION><CLASS>tier1</CLASS></DESTINATION></TO>
  <WHEN>
    <IOTEMP Type="nrwbytes "><MIN Flags="gt">5</MIN><PERIOD>2</PERIOD></IOTEMP>
  </WHEN>
</RELOCATE>
```

I/O temperature is a measure of data transfer activity relative to file size. This rule relocates a file only if it has experienced average daily I/O activity greater than five times its size during the 48 hour period immediately preceding policy enforcement. This rule might be used to return files that had been relocated to `tier2` volumes due to inactivity back to `tier1` volumes when application activity against resumes. Using I/O temperature rather than access age as the relocation qualifier reduces the chance of relocating files that are not actually being used frequently by applications.

RELOCATE statement example: multiple destinations

By specifying multiple `<DESTINATION>` elements in the `<TO>` clause of a `RELOCATE` statement, an administrator can direct VxFS to relocate files to a secondary placement class if adequate space is not available on the primary class. Fragment 3-12 illustrates a rule for relocating files with low I/O temperatures from `tier1` volumes to `tier2` volumes, and to `tier3` volumes when `tier2` volumes are fully occupied.

Fragment 3-12 Two-tier RELOCATE statement

```
<RELOCATE>
  <FROM><SOURCE><CLASS>tier1</CLASS></SOURCE><FROM>
  <TO> [3]
    <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
```

Fragment 3-12 Two-tier RELOCATE statement *(continued)*

```
    <DESTINATION><CLASS>tier3</CLASS></DESTINATION>
  </TO>
  <WHEN>
    <IOTEMP Type="nrbytes"><MAX Flags="lt">4</MAX><PERIOD>3</PERIOD></IOTEMP>
  </WHEN>
</RELOCATE>
```

The rule in Fragment 3-12 relocates qualifying files (those whose 3-day average I/O temperatures are less than 4, as specified in the `<WHEN>` clause) that reside on `tier1` volumes as it encounters them in its scan of the file system directory tree. When it calculates that already-relocated files would completely fill `tier2`, VxFS relocates qualifying files to `tier3` instead.

RELOCATE statement example: specified source locations

The optional `<FROM>` clause in the `RELOCATE` statement restricts the source placement classes from which files are relocated. If a `RELOCATE` statement does not contain a `<FROM>` clause, VxFS relocates qualifying files selected by the rule no matter where they reside when the policy is enforced.

Fragment 3-13 illustrates a part of a policy rule that relocates files according to their sizes, no matter what volumes they reside on when the policy is enforced.

Fragment 3-13 RELOCATE statements with no <FROM> clauses

```
[1]       <RELOCATE>
[2]         <TO><DESTINATION><CLASS>tier1</CLASS></DESTINATION></TO>
[3]         <WHEN><SIZE Units=MB><MAX Flags="lt">10</MAX></SIZE></WHEN>
[4]       </RELOCATE>

[5]       <RELOCATE>
[6]         <TO><DESTINATION><CLASS>tier2</CLASS></DESTINATION></TO>
[7]         <WHEN>
[8]           <SIZE Units=MB><MIN Flags="gteq">10</MIN><MAX Flags="lt">100</MAX></SIZE>
[9]         </WHEN>
[10]      </RELOCATE>

[11]     <RELOCATE>
[12]        <TO><DESTINATION><CLASS>tier3</CLASS></DESTINATION></TO>
[13]        <WHEN><SIZE Units=MB><MIN Flags="gteq">100</MIN></SIZE></WHEN>
[14]      </RELOCATE>
```

The `RELOCATE` statement in lines `[1]-[4]` of Fragment 3-13 relocates files smaller than 10 megabytes to `tier1` volumes. The statement in lines `[5]-[10]` relocates files of between 10 and 100 megabytes to `tier2` volumes, and the statement in lines `[11]-[14]` relocates files larger than 100 megabytes to `tier3` volumes. When the policy is enforced, VxFS relocates all qualifying files that do not already reside in their destination placement classes.

The DELETE statement

The `DELETE` statement is very similar to the `RELOCATE` statement in both form and function, lacking only the `<TO>` clause. File placement policy-based deletion may be thought of as relocation to the null device. As with any irreversible operation on data, it is good practice to back up data prior to enforcing file placement policies that contain `<DELETE>` statements.

Fragment 3-14 shows the general form of the `DELETE` statement.

Fragment 3-14 General form of the DELETE statement

```
[1]      <DELETE>
[2]        <FROM>
[3]          <SOURCE>
[4]            <CLASS>...placement_class_name...</CLASS>
[5]          </SOURCE>
[6]          <SOURCE>...additional placement class specifications...</SOURCE>
[7]        </FROM>
[8]        <WHEN>...deletion conditions...</WHEN>
[9]      </DELETE>
```

As Fragment 3-14 indicates, a `DELETE` statement contains the two clauses listed in Table 3-5.

Table 3-5 Clauses in the DELETE statement

Clause	Description
`<FROM>`	An optional clause (lines `[2]-[7]`) that contains a list of placement classes from which VxFS deletes files if they meet the qualifying conditions specified in the `<WHEN>` clause. No priority is associated with the ordering of placement classes in a `<FROM>` clause. If a qualifying file selected by the rule is located in any specified placement class, VxFS deletes it. If a `DELETE` statement does not contain a `<FROM>` clause, VxFS deletes qualifying files no matter which volumes they reside on.

Table 3-5 Clauses in the DELETE statement *(continued)*

Clause	Description
`<WHEN>`	An optional clause (line `[8]`) indicating the conditions under which files selected by the rule should be deleted. The form of the `<WHEN>` clause in a `DELETE` statement is identical to that of the `<WHEN>` clause in a `RELOCATE` statement (Fragment 3-8). If a `DELETE` statement does not contain a `<WHEN>` clause, VxFS deletes all files selected by the rule. If a `<FROM>` clause is present, VxFS deletes files only from the placement classes it specifies.

VxFS enforces placement policies file by file. Only the first rule that selects a file applies to it. Similarly, within a rule, only the first statement for which a file qualifies applies to it. Thus, if a file is selected by two different rules, whichever rule occurs first in the policy applies to it, even if the rule results in no action. Similarly, if a file qualifies for relocation and deletion under the same rule, whichever statement occurs first in the policy will be the operative one.

DELETE statement examples

Fragment 3-15 shows two examples of `DELETE` statements.

Fragment 3-15 Sample DELETE statements

```
[1]      <DELETE>
[2]        <FROM><SOURCE><CLASS>tier3</CLASS></SOURCE></FROM>
[3]      </DELETE>

[4]      <DELETE>
[5]        <FROM><SOURCE><CLASS>tier2</CLASS></SOURCE></FROM>
[6]        <WHEN><ACCAGE Units=days><MIN Flags=gt>120</MIN></ACCAGE></WHEN>
[7]      </DELETE>
```

The first `DELETE` statement (lines `[1]`-`[3]`) in Fragment 3-15 deletes files that reside on `tier3` volumes when the policy is enforced. The absence of a `<WHEN>` clause in the `DELETE` statement indicates that deletion of files selected by the rule is unconditional. This type of rule might be used, for example, when an entire storage tier is to be retired.

The second `DELETE` statement (lines `[4]`-`[7]`) deletes files selected by the rule that reside on `tier2` volumes when policy is enforced, provided that they have not been accessed for the 120 days immediately prior to policy enforcement.

Calculating a file’s I/O temperature and access temperature

One important application of the VxFS Dynamic Storage Tiering facility is automating the relocation of inactive files to lower-cost storage. On the surface, this seems simple enough. If a file has not been accessed for a certain period of time (specified in the `<ACCAGE>` element introduced in Fragment 3-7), enforcement of the file system’s placement policy should relocate it to a lower storage tier.

But access age is inadequate as the only qualifier for activity-based relocation for the two reasons listed in Table 3-6.

Table 3-6 Shortcomings of access age-based file relocation

Shortcoming	Description
It is a binary measure	The access age of a file is computed by subtracting the time at which policy enforcement starts from the file’s POSIX `atime`. If a file is opened the day before the policy is enforced, its access age is one day, even though it may have been inactive for the month preceding. If the intent of a policy rule is to relocate inactive files to lower-tier volumes, it will perform badly with files that are accessed, however casually, within the interval specified by the value of the `<ACCAGE>` element.
It is not the best indicator of resumption of significant activity	Using access age as the qualifier for relocating inactive files to lower-tier volumes may fail to schedule some relocations that should be performed, but at least it results in less relocation activity than necessary. Using `ACCAGE` as the qualifier for relocating previously inactive files that have become active is likely to schedule unwarranted relocation activity. For example, a rule that specifies relocation of files on `tier2` that have been accessed within the last three days to `tier1`, no distinction is made between a file whose ownership was changed and one that was read or written intensively by applications.

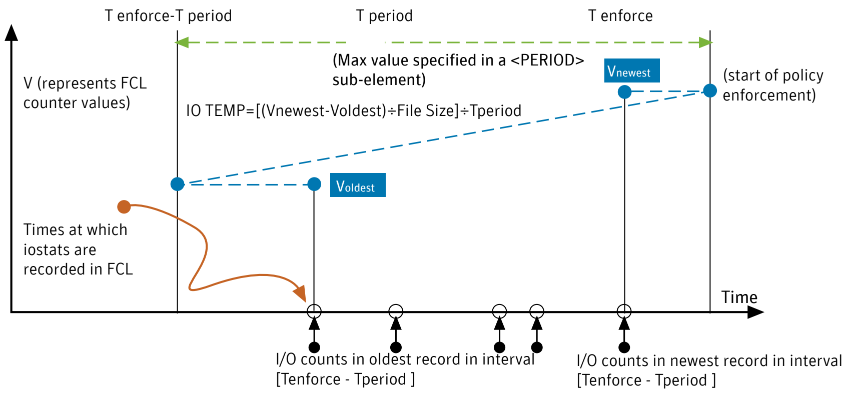
The VxFS Dynamic Storage Tiering facility implements the concepts of average I/O temperature and average access temperature to overcome these deficiencies. A file’s I/O temperature is calculated as the number of bytes transferred to or from it over a specified period of time divided by its size. For example, if a file occupies one megabyte of storage at the time of policy enforcement, and the data in it has been completely read or written 15 times within the last three days, VxFS calculates its 3-day average I/O temperature to be 5 (15 megabytes of I/O ÷ 1 megabyte file size ÷ 3 days).

Similarly, a file’s average access temperature is the number of read or write requests made to it over a specified number of 24-hour periods divided by the number of periods. Unlike I/O temperature, access temperature is unrelated to file size; a large file to which 20 I/O requests are made over a 2-day period has the same average access temperature as a small file accessed 20 times over a 2-day period.

If a file system's active placement policy includes any `<IOTEMP>` or `<ACCESSTEMP>` clauses, VxFS begins policy enforcement by using information in the file system's File Change Log (FCL) to calculate average I/O activity against all files in the file system during the longest `<PERIOD>` specified in the policy (if shorter periods are specified, they are ignored). It uses these calculations to qualify files for I/O temperature-based relocation and deletion.

Figure 3-2 depicts the I/O temperature computation graphically.

Figure 3-2 I/O temperature computation


T enforce-T period
T period
T enforce
(Max value specified in a <PERIOD> sub-element)
V (represents FCL counter values)
Vnewest
(start of policy enforcement)
IO TEMP=[(Vnewest-Voldest)÷File Size]÷Tperiod
Voldest
Times at which iostats are recorded in FCL
Time
I/O counts in oldest record in interval [Tenforce - Tperiod]
I/O counts in newest record in interval [Tenforce - Tperiod]

As its name implies, the File Change Log records information about changes made to files in a VxFS file system. In addition to recording creations, deletions, extensions, and so forth, the FCL can be configured to periodically capture the cumulative amount of I/O activity (number of bytes read and written and number of reads and writes) against active files. When configured to collect I/O statistics, VxFS records file I/O activity counters in the FCL each time a file's inode enters or leaves the VxFS inode cache, as well as at user-specifiable intervals when a file is actually experiencing I/O activity. (The latter captures periodic information about files that remain open for long periods).

The period of interest in I/O temperature calculations is the interval between the time at which enforcement occurs (*T*enforce in Figure 3-2) and that time minus the largest interval specified in any `<PERIOD>` element in the active policy (*T*period in Figure 3-2).

The I/O temperature calculation

For files with I/O activity during *T*period, VxFS computes an approximation of the amount of data read, data written, and total data transfer (the sum of data read and data written), effectively by subtracting the I/O levels in the file's oldest FCL I/O statistics record within the period of interest from those in the file's newest record (*V*newest - *V*oldest in Figure 3-2). It then computes each file's total I/O temperature by dividing its I/O activity by its size at *T*enforce. Average I/O temperature for the period is computed by dividing total I/O temperature by *T*period. Dividing by file size normalizes the result, so that a file's temperature is proportional to its size. I/O temperature is the ratio of the number of bytes of I/O experienced by a file during the period to the file's size. If data transfer activity is not the primary consideration, `<ACCESSTEMP>` is a more appropriate qualifier, because it measures file access events without regard to the amount of data transferred.

While these computations are approximations in several ways, they represent easy to compute, and more important, unbiased, estimates of relative recent I/O activity upon which relocation decisions can be based.

File relocation based on <IOTEMP>

File relocation and deletion decisions can be based on read, write, or total I/O activity. Fragment 3-16 illustrates the use of `IOTEMP` in a policy rule to specify relocation of low-activity files from `tier1` volumes to `tier2` volumes.

Fragment 3-16 Sample IOTEMP clause

```
<RELOCATE>
  <FROM><SOURCE><CLASS>tier1</CLASS></SOURCE></FROM>
  <TO><DESTINATION><CLASS>tier2</CLASS></DESTINATION></TO>
  <WHEN>
    <IOTEMP Type="nrwbytes "><MAX Flags="lt">3</MAX><PERIOD>4</PERIOD></IOTEMP>
  </WHEN>
</RELOCATE>
```

Fragment 3-16 specifies that files selected by the rule should be relocated from `tier1` volumes to `tier2` volumes if their average I/O temperatures fall below `3` over a period of `4` days. The `Type=nrwbytes` attribute specifies that total data transfer activity (the sum of bytes read and bytes written) is to be used in the computation. For example, a 50 megabyte file that experienced less than 150 megabytes of data transfer per day over the 4-day period immediately preceding policy enforcement would have an I/O temperature less than 3, and would therefore be relocated, whereas a 40 megabyte file with the same amount of I/O activity would have an I/O temperature of 3.75, and would not be relocated. A file that experiences no activity during a period of interest has an I/O temperature of zero for the period. VxFS relocates qualifying files in the order in which it encounters them as it scans the file system directory tree.

Using average I/O temperature or access temperature rather than a binary indication of activity (POSIX `atime` or `mtime`) minimizes the chances of failing to relocate files that are nearly inactive during the period of interest. A large file that experiences only a few bytes of I/O or a few requests during the period would have a low temperature, and would therefore become a candidate for relocation to `tier2` volumes, even if the activity was recent.

A possibly more important use of I/O temperature-based file relocation is upward relocation—the relocation of files previously placed on lower storage tiers because of inactivity or low temperature back to higher tiers in the hierarchy. Fragment 3-17 shows a sample `RELOCATE` statement for relocating files from `tier2` volumes to `tier1` when read activity level against them increases.

Fragment 3-17 Using IOTEMP for upward relocation of active files

```
<RELOCATE>
  <FROM><SOURCE><CLASS>tier2</CLASS></SOURCE></FROM>
  <TO><DESTINATION><CLASS>tier1</CLASS></DESTINATION></TO>
  <WHEN>
    <IOTEMP Type="nrbytes">
      <MAX Flags="gt">5</MAX><PERIOD>2</PERIOD>
    </IOTEMP>
  </WHEN>
</RELOCATE>
```

The `RELOCATE` statement in Table 3-6 relocates files on `tier2` volumes whose two-day average I/O temperatures (calculated using the number of bytes read) are greater than `5` to `tier1` volumes. Bytes written to the file during the two-day period of interest do not contribute to the calculation.

Using average I/O temperature or access temperature rather than access age as a file relocation qualifier gives administrators a precise control over automated file relocation that can be used to adjust file placement policies to application requirements. For example, a larger maximum `<PERIOD>` element in a rule tends to prevent files from being relocated unless I/O activity against them is sustained over that period. Alternatively, specifying a higher temperature and a shorter maximum period tends to relocate files based on short-term I/O intensity, ignoring activity that is further in the past.

Multiple selection criteria and qualifiers in file placement policy rule statements

There are four cases in which file placement policy statements may contain multiple clauses of the same type. The sections that follow describe policy behavior in these four cases.

Case 1: multiple file selection criteria in SELECT statement clauses

Within a single `SELECT` statement, all the selection criteria clauses of a single type are treated as a selection list. A file need only satisfy a single criterion of a given type to be designated. For example, in Fragment 3-18, files in any of the `db/datafiles`, `db/indexes`, and `db/logs` directories (all relative to the file system mount point) would be selected.

Fragment 3-18 SELECT statement specifying multiple <DIRECTORY> criteria

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">db/datafiles</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/indexes</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/logs</DIRECTORY>
</SELECT>
```

When a `SELECT` statement includes multiple types of file selection criteria, a file must satisfy one criterion of each included type in order for the rule's action statements to apply to it. For example, in the `SELECT` statement in Fragment 3-19, a file must reside in one of `db/datafiles`, `db/indexes`, or `db/logs` and be owned by one of `DBA_Manager`, `MFG_DBA`, or `HR_DBA` in order to be designated for possible action.

Fragment 3-19 SELECT Statement specifying multiple selection criteria types

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">db/datafiles</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/indexes</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/logs</DIRECTORY>
  <USER>DBA_Manager</USER>
  <USER>MFG_DBA</USER>
  <USER>HR_DBA</USER>
</SELECT>
```

If a rule includes multiple `SELECT` statements, a file need only satisfy one of them to be selected by the rule. This property can be used to specify alternative conditions for file selection. For example, in the `SELECT` statement in Fragment 3-20, a file need only reside in one of `db/datafiles`, `db/indexes`, or db/logs or be owned by one of DBA_Manager, MFG_DBA, or HR_DBA in order to be selected by the rule. It need not satisfy both criteria.

Fragment 3-20 Multiple SELECT statements

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">db/datafiles</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/indexes</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/logs</DIRECTORY>
</SELECT>

<SELECT>
  <USER>DBA_Manager</USER>
  <USER>MFG_DBA</USER>
  <USER>HR_DBA</USER>
</SELECT>
```

Case 2: multiple placement classes in CREATE statement <ON> clauses and in RELOCATE statement <TO> clauses

Both the <ON> clause of the CREATE statement and the <TO> clause of the RELOCATE statement can specify priority ordered lists of placement classes by using multiple <DESTINATION> elements. VxFS creates or relocates files in the first placement class listed if possible. If the first listed class has insufficient free space (or if the file system's volume set does not include any volumes in that class), VxFS uses the second listed class. If the second listed class has insufficient free space, VxFS tries the third listed class, and so forth. Fragment 3-21 gives an example of three placement classes specified in the <ON> clause of a CREATE statement.

Fragment 3-21 Multiple placement classes in a CREATE statement <ON> clause

```
<CREATE>
<ON>
<DESTINATION><CLASS>tier1</CLASS></DESTINATION>
<DESTINATION><CLASS>tier2</CLASS></DESTINATION>
<DESTINATION><CLASS>tier3</CLASS></DESTINATION>
</ON>
</CREATE>
```

In the example of Fragment 3-21, VxFS allocates space for newly-created files selected by the rule on tier1 volumes if space is available. If no tier1 volume has sufficient free space, VxFS attempts to allocate space on a tier2 volume. If no tier2 volume has sufficient free space, VxFS attempts allocation on a tier3 volume. If VxFS cannot allocate sufficient space for the new file in any of the three specified placement classes, allocation would fail with an ENOSPC error indication, even if the file system's volume set included other placement classes that did have sufficient space. (The Flags=any attribute can be specified in the <ON> tag to cause

VxFS to allocate space elsewhere in the file system's volume set if all of the listed placement classes are fully occupied.)

The `<TO>` clause in the `RELOCATE` statement behaves similarly. VxFS relocates qualifying files to the first placement class specified if space permits, to the second specified class if not, and so forth. If none of the destination placement classes have available space, qualifying files are not relocated, but no error is signaled.

Case 3: multiple placement classes in <FROM> clauses of DELETE and RELOCATE statements

The `<FROM>` clause in `RELOCATE` and `DELETE` statements can include multiple source placement classes. Unlike the `<ON>` and `<TO>` clauses, no priority is implied by the order in which `<SOURCE>` elements appear in `<FROM>` clauses. If a qualifying file resides in any of the placement classes specified in a `<FROM>` clause, VxFS deletes or relocates it, space permitting.

Case 4: multiple qualifiers in <WHEN> clauses of RELOCATE and DELETE statements

The `<WHEN>` clause in a `DELETE` or `RELOCATE` statement may include multiple qualifications. Each of `<ACCAGE>`, `<MODAGE>`, `<SIZE>`, `<IOTEMP>`, and `<ACCESSTEMP>` may be specified once in a single clause. When multiple qualifications are specified, all must be satisfied in order for a selected file to be relocated or deleted. For example, in the `<WHEN>` clause in Fragment 3-22, a selected file would have to be both inactive (not accessed) for more than 30 days and larger than 100 megabytes to be eligible for relocation or deletion.

Fragment 3-22 Multiple qualifiers in a <WHEN> clause

```
<WHEN>
  <ACCAGE Units="days"><MIN Flags="gt">30</MIN></ACCAGE>
  <SIZE Units="MB"><MIN Flags="gt">100</MIN></SIZE>
</WHEN>
```

Because an active policy enforcement scan against a given file ends at the first rule that selects the file, and because a `RELOCATE` or `DELETE` statement may contain only one `<WHEN>` clause, it is not possible to write rules to relocate or delete files that meet any one of two or more relocation or deletion qualifiers.

File placement policy rule and statement ordering

Administrators can use the Dynamic Storage Tiering graphical console to create and assign file placement policies of any of four built-in types. Alternatively, they can use text editors or XML editors to create XML placement policy documents, and assign the policies to file systems using the `fsppadm assign` command. The graphical interface orders rule statements for the policies it creates in the correct order to achieve the behavior described in Chapter 6 . With policies created using a text or XML editor, the creator is responsible for ordering rules and the statements in them correctly.

The rules comprising a policy may occur in any order, but during both initial file allocation and policy enforcement, the first rule that selects a file is the only rule that applies to that file. Thus, rules whose purpose is to supersede a general behavior for a certain set of files must precede the more general rules in a file placement policy if they are to be effective. To illustrate this point, shows an example of faulty rule placement with potentially unintended consequences.

Fragment 3-23 Example of ineffective rule ordering

```
<RULE Name=GeneralRule>
  <SELECT><PATTERN>*</PATTERN></SELECT>
  <CREATE><ON><DESTINATION><CLASS>tier2</CLASS></DESTINATION></ON></CREATE>
...other statements...
</RULE>

 <RULE Name=DatabaseRule >
  <SELECT><PATTERN>*.db</PATTERN></SELECT>
  <CREATE><ON><DESTINATION><CLASS>tier1</CLASS></DESTINATION></ON></CREATE>
...other statements...
</RULE>
```

The `GeneralRule` in Fragment 3-23 specifies that all new files created in the file system (specified by `<PATTERN>*</PATTERN>`) are to be created on `tier2` volumes. The `DatabaseRule` specifies that files whose names have an extension of `db` following the period are to be created on `tier1` volumes. The `GeneralRule` applies to all files created in the file system, including those whose names have an extension of `db` following the period, so `DatabaseRule` never applies to any file. This fault can be remedied by exchanging the order of the two rules. If `DatabaseRule` occurs first in the policy, VxFS encounters it first when determining where to place new files whose names follow the pattern `*.db`, and correctly allocates space for them on `tier1` volumes. For files to which `DatabaseRule` does not apply, VxFS continues scanning the policy and allocates `tier2` space according to the specification in the `CREATE` statement of `GeneralRule`.

A similar consideration applies to the statements within a policy rule. When it is enforcing a policy against a file, VxFS scans the statements within a rule in order of their occurrence until it encounters an applicable one. This can result in unintended behavior. For example, Fragment 3-24 illustrates a `RELOCATE` statement and a DELETE statement in a rule that is intended to relocate if they have not been accessed in 30 days, and delete them if they have not been accessed in 90 days.

Fragment 3-24 Misplaced statements in a file placement policy rule

```
[1]     <RELOCATE>
[2]       <TO><DESTINATION><CLASS>tier2</CLASS></DESTINATION></TO>
[3]       <WHEN><ACCAGE Units="days"><MIN Flags="gt">30</MIN></ACCAGE>
[4]     </RELOCATE>

[5]     <DELETE>
[6]       <WHEN><ACCAGE Units="days"><MIN Flags="gt">90</MIN></ACCAGE>
[7]     </DELETE>
```

If the rule is written as in Fragment 3-24, with the RELOCATE statement preceding the DELETE statement, files are never deleted, because the <WHEN> clause in the RELOCATE statement applies to all selected files that have not been accessed for at least 30 days, including those that have not been accessed for 90 days. VxFS ceases its enforcement scan against a file when it identifies a statement for which the file qualifies, so deletion based on the rule in lines [5]-[7] can never occur. This example illustrates the general principle that RELOCATE and DELETE statements specifying less inclusive qualifications should precede statements that specify more inclusive ones. Symantec-supplied graphical interfaces for creating built-in policies automatically produce the correct statement order for the policies they create.

File placement policies and extending files

In a VxFS file system with an active file placement policy, the placement class of the volume on which a file resides is part of the file's metadata, attached when the file is created, and updated whenever VxFS relocates it. When an application extends a file, VxFS allocates space for the extension on the volume occupied by the file if possible, and on another volume in the same placement class if not. For example, if a file is created on a tier1 volume, extensions that occur prior to any relocation are also on a tier1 volume. If the file is later relocated to a tier2 volume, extensions that occur after the relocation are also in tier2.

Summary: key properties of multi-volume file systems

The VxFS Dynamic Storage Tiering (DST) facility is an easy-to-implement mechanism for achieving the reduced average hardware cost and quality of service improvements inherent in multi-tier online storage without incurring excessive offsetting administrative cost. A key benefit resulting of Dynamic Storage Tiering is the ability to place files on the best type of storage without the need to manage a separate file system for each tier. With Dynamic Storage Tiering, correct file placement occurs automatically at the three levels listed in Table 3-7.

Table 3-7 Automatic online storage administration with the Dynamic Storage Tiering facility

Level	Description
Creation	DST placement policy rules automatically ensure that newly created files are placed on the right type of storage, whether right is defined either by business requirements such as criticality to the enterprise or by technical ones such as need for high transactional I/O performance.
Changing circumstances	DST file placement policy rules automatically relocate files to new right storage devices when their properties (size, access frequency, I/O temperature, ownership, or logical position in the name space) change, without administrative intervention. Moreover, as storage capacity requirements grow and volumes are added to a file system's volume set, DST automatically applies its active placement policy to them, maintaining correct placement according to policy within the expanded volume set, again, without administrative intervention.
Across file systems	Because DST placement policies specify placement classes rather than specific volumes, a single general policy can be assigned to any number of hosts that have similar data management requirements and hardware configurations. This promotes standardization across the data center and the enterprise, and tends to result in fewer administrative errors.

Dynamic Storage Tiering provides the twin keys to successful exploitation of multi-tier storage—flexibility and automation. VxFS administrators can define a variety of rules for placing and relocating files within multiple storage tiers. Automation makes relocation unobtrusive, both in terms of administrative effort and in terms of real-time performance impact on applications.

Chapter 4

Using the Storage Foundation Dynamic Storage Tiering facility

This chapter includes the following topics:

Implementing Dynamic Storage Tiering

Use of the Storage Foundation Dynamic Storage Tiering facility requires that file systems and volumes be properly prepared for it. File systems must be multi-volume file systems, whose volume sets consist of volumes that are tagged with the placement classes named in any policies that will be assigned to them. These preparations can be made using Storage Foundation Command Line Interpreter (CLI) commands issued on the host that controls the target file system and its volume set, or they can be performed centrally by the Storage Foundation Management Server on behalf of systems with Management Server client software installed.

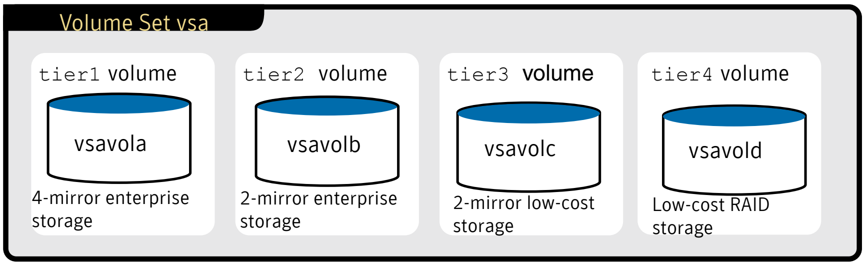
This chapter describes the use of Storage Foundation Command Line Interpreter (CLI) commands to create a multi-volume VxFS file system, assign a Dynamic

Storage Tiering file placement policy to it, and manage file placement within the file system.

Chapter 11 describes how to convert a single-volume VxFS file system into a multi-volume one.

Figure 4-1 illustrates four virtual volumes that might be made into a volume set to house a file system providing four levels of service.

Figure 4-1 Volumes for a multi-volume file system



The volume, `vsavola`, consists of four mirrors, perhaps constructed from two 2-mirror LUNs presented by two disk arrays. The `vsavolb` volume is built on a single 2-mirror LUN that is protected against disk failure but not against complete disk array failure. The `vsavolc` volume is also built on a 2-mirror LUN, but one presented by a lower-cost mid-range disk array. The `vsavold` volume is also built on a mid-range disk array LUN, but in this case, a parity RAID-5 one for still lower-cost failure protection.

These volumes can be created using any of the three VxVM volume creation mechanisms: administrator-directed configuration, the `vxassist` utility, or intelligent provisioning storage templates. Each is optimized for a specific purpose. The `vsavola` volume is configured for highest availability, and would presumably be used to store its hosting system's most critical data. The `vsavolb` and `vsavolc` volumes can tolerate disk failure, but not complete failure of the arrays that present them. The primary difference between the two is likely to lie in the I/O performance they deliver. The `vsavold` volume tolerates fewer failure modes, but provides more economical storage, and would probably be most useful for storing online archives of infrequently-updated data with minimal application performance implications.

Tagging volumes

The first step in preparing to use Dynamic Storage Tiering is to tag these volumes so that they are distinguishable by DST. VxVM can assign one or more character strings, called tags, to a volume. Storage Foundation as well as other applications can use volume tags to represent extrinsic volume properties.

VxVM organizes volume tags as a hierarchical name space in which the levels of the hierarchy are separated by period characters. By convention, the uppermost level in the volume tag hierarchy denotes the Storage Foundation component or application that uses a tag, the second level denotes the tag's purpose, and the third level is the tag's actual value.

For example, the VxFS Dynamic Storage Tiering facility recognizes volume tags of the form `vxfs.placement_class.class_name`. The prefix `vxfs` identifies a tag as being used by the VxFS file system. The `placement_class` part identifies the tag as a file placement class recognized by the Dynamic Storage Tiering facility. The `class_name` part represents the user-defined name of the file placement class to which the tagged volume belongs. For example, a volume having the tag `vxfs.placement_class.tier1` is said to belong to placement class `tier1`. Only the `tier1` part of the volume tag appears in file placement policies.

Administrators use the `vxvoladm volume settag` command to associate tags with volumes, as Dialog 4-1 illustrates.

Dialog 4-1 Setting volume tags for Dynamic Storage Tiering facility use

```
[1]      # vxvoladm -g dstdg settag vsavola vxfs.placement_class.tier1
[2]      # vxvoladm -g dstdg settag vsavolb vxfs.placement_class.tier2
[3]      # vxvoladm -g dstdg settag vsavolc vxfs.placement_class.tier3
[4]      # vxvoladm -g dstdg settag vsavold vxfs.placement_class.tier4
```

VxFS policy rules specify file placement in terms of placement classes rather than in terms of specific volumes. All volumes that belong to a particular placement class are interchangeable with respect to placement policy operations. VxFS balances space allocation approximately equally across all the volumes in a placement class. Specifying file placement in terms of placement classes rather than in terms of specific volumes simplifies multi-tier storage administration in the two important ways listed in Table 4-1.

Table 4-1 Advantages of file placement classes over volumes

Advantage	Description
Adding or removing volumes does not require a file placement policy change.	If a volume with a tag value of `vxfs.placement_class.tier2` is added to a file system's volume set, all policies that refer to `tier2` immediately apply to the newly-added volume with no administrative action. Similarly, volumes can be evacuated (have data removed from them) and removed from a file system without a policy change. The active policy continues to apply to the file system's remaining volumes.
File placement policies are not specific to individual file systems.	Using the Storage Foundation Management Server, a single file placement policy can be assigned to any file system in a data center whose volume set includes volumes tagged with the placement classes named in the policy. This property makes it possible for enterprises with large numbers of servers to define standard placement policies and apply them uniformly to many servers with a single administrative action.

Creating volume sets and multi-volume file systems

Once volumes are tagged for Dynamic Storage Tiering use, a volume set can be created and a multi-volume file system can be formatted on it. Dialog 4-2 illustrates the creation of a volume set called `vsa` using `vsavola` as the first volume, called volume 0, and the addition of `vsavolb`, `vsavolc`, and `vsavold` to it.

Dialog 4-2 Creating a volume set and a multi-volume file system

```
[1]  # vxvset -g dstdg  make vsa vsavola
[2]  # vxvset -g dstdg addvol vsa vsavolb
[3]  # vxvset -g dstdg addvol vsa vsavolc
[4]  # vxvset -g dstdg addvol vsa vsavold

[5]  # mkfs -F vxfs /dev/vx/rdsk/dstdg/vsa
[6]  # mkdir /vsa
[7]  # mount -F vxfs /dev/vx/dsk/dstdg/vsa /vsa /vsa
```

The configuration of volume 0 in a volume set is critical because it is where VxFS stores key file system metadata structures. For this reason, volume 0 cannot be designated `dataonly` or ineligible to contain metadata. Because volume 0 must be present in order for a file system to be mounted, it should be a failure-tolerant volume. By default, VxFS stores all file system metadata on volume 0, and treats all other volumes in its volume set as `dataonly`. The eligibility of any volume other than volume 0 to store metadata can be changed by using the `fsvoladm clearflags` command.

From an administrator's standpoint, a volume set resembles a single volume in most respects. It appears as a single volume in the output of the `vxprint` command, for example. To create a multi-volume file system, an administrator uses the POSIX `mkfs` command, specifying device path name that leads to a volume set as the command target (line `[5]`).

Policies that use I/O temperature as a file relocation qualifier

The next step in preparing to use the Dynamic Storage Tiering facility depends upon whether the policy that will be assigned to the file system uses average I/O temperature or access temperature (defined Relocation qualifiers) as a relocation or deletion qualification. VxFS stores the file activity statistics used in `IOTEMP` and `ACCESSTEMP` calculations in its File Change Log (FCL). To minimize overhead in file systems that do not use the statistics, VxFS does not store them in the FCL by default. When a file placement policy containing `IOTEMP` or `ACCESSTEMP` qualifiers is assigned to a file system, VxFS automatically starts the FCL and causes statistics to be stored in it. Dialog 4-3 shows the maximal set of VxFS responses to an assignment of a policy containing temperature-based deletion or relocation statements.

Dialog 4-3 Enabling File Change Log statistics collection

```
[1]      # fsppadm assign /dst /opt/FSPROtest/bin/iotemp.xml
[2]      UX:fsppadm:Warning: V-175-50999-159: File Change Log was not on for /dst
[3]      UX:fsppadm:Warning: V-175-50999-164: File Change Log has been turned on for /dst
[4]      UX:fsppadm:Warning: V-175-34615-185: File Change Log filestats set for /dst
[5]      UX:fsppadm:Warning: V-175-34615-205: The fcl_keeptime for /dst is 2 days, policy
[6]      requires 5 days of filestats, Some IOTEMP based reloction and deletion may not occur
[7]      # vxtunefs -o fcl_keeptime=432000 /dst
```

The warnings in lines `[2]-[4]` of Dialog 4-3 simply indicates that VxFS discovered that FCL records were not being collected at the time of policy assignment, and responded by enabling the FCL and directing that `filestats` be stored in it. The warning in lines `[5]-[6]` indicates that the current setting of the VxFS tunable `fcl_keeptime`, which determines how long FCL records are retained, is inadequate for the requirements of the policy being assigned. VxFS does not automatically adjust `fcl_keeptime`.

The `vxtunefs` command on line `[7]` increases the maximum number of seconds for which VxFS retains I/O statistics and other records in the FCL. The example specifies retention of five days (432,000 seconds). The `fcl_keeptime` should be set at or slightly larger than the largest value specified in any `<PERIOD>` element in the policy.

File Change Log considerations

The storage space used by an FCL depends primarily upon the number of file change events during the `fcl_keeptime`, and must ultimately be determined experimentally. Typically, however, policies that include `IOTEMP` or `ACCESSTEMP` as a relocation qualifier result in substantially larger `fcl_keeptime` values than other applications of the FCL such as backup. Since `fcl_keeptime` applies to all FCL records, not just I/O statistics, an FCL's size is roughly proportional to the length of time for which records are kept, at least in a file system whose activity pattern is relatively constant. The space required by an FCL when a temperature-based policy is in effect can be estimated by recording the FCL size and the `fcl_keeptime` value before file placement policy assignment, and multiplying the pre-assignment FCL size by the ratio of the post- and pre-assignment `fcl_keeptime`s. For example, if an FCL occupies 10 megabytes with a pre-assignment `fcl_keeptime` of 1 day, and the post-assignment `fcl_keeptime` is 8 days, the FCL can be expected to grow to 80 megabytes with the policy in effect.

The growth of an FCL that may be required by temperature-based policies can be particularly significant because VxFS treats FCLs as metadata, meaning that they cannot be stored on `dataonly` volumes. A file system's `metadataok` storage capacity should be increased if necessary when a policy containing `<IOTEMP>` or `<ACCESSTEMP>` qualifiers is assigned.

During enforcement of a policy that contains `<IOTEMP>` or `<ACCESSTEMP>` qualifiers, VxFS creates a file called `.__fsppadm_fcliotemp.db` in the file system's `lost+found` directory. If the size of the file is less than 50 megabytes, it remains after enforcement for possible use by `fsppadm query` operations, otherwise it is deleted. Each policy enforcement deletes the `.__fsppadm_fcliotemp.db` file if it is present and creates a new one.

Temperature-based relocation and deletion is not possible until the FCL has been running at least as long as the longest `<PERIOD>` specified in any `RELOCATE` or `DELETE` statement. It may occasionally be desirable for an administrator to start the FCL and enable statistics collection prior to policy enforcement using a command sequence similar to that shown in Dialog 4-4.

Dialog 4-4 Enabling File Change Log statistics collection

```
[1]      # fcladm on /dst
[2]      # fcladm set filestats /dst
```

The `fcladm on` command in line `[1]` of Dialog 4-4 enables basic File Change Log data collection on the file system mounted at `/dst`. I/O activity statistics are not stored persistently in the FCL by default, so the command in line `[2]` is required to store them (`filestats`) for the `/dst` file system. When persistent `filestats` storing is enabled for a file system, data about I/O activity is appended to certain key event records recorded in the FCL, and additional statistics-only records are recorded periodically.

If a file system's active file placement policy does not include `IOTEMP` or `ACCESSTEMP` as a relocation qualifier, the Dynamic Storage Tiering facility does not require FCL and persistent storage of file I/O statistics to be enabled.

Assigning file placement policies

When its volumes have been tagged with placement classes and its FCL has been tuned (if necessary), a multi-volume file system is ready for use by the Dynamic Storage Tiering facility. To use DST with such a file system, an administrator creates a file placement policy using the grammar described in Chapter 3 , and assigns it to the file system using the `fsppadm assign` command in a fashion similar to that shown in Dialog 4-5. Alternatively, facilities of the Veritas Enterprise Administrator (VEA) graphical management console can be used to create one of the four built-in policies described in Chapter 6. The VEA console's policy creation wizards automatically assign each policy they create as the active policy for the file system in whose context they are operating.

Dialog 4-5 Analyzing a file system with an active file placement policy

```
# fsppadm assign /vsa /tmp/policy.xml
                           ... applications use file system...
# fsppadm analyze -l /vsa
/vsa/NODE0001/SQL00001/SQL1.DAT                             Database_class1
                                               Will be relocated to - tier1
/vsa/suzi/inst01/NODE0000/SQL00001/SQL1.DAT                Database_class1A
                                               Will be relocated to - tier1
/vsa/suzi/inst01/NODE0000/SQL00001/SQL11.DAT               Database_class1A
                                               Will be relocated to - tier1
Files (to be) moved : 3

Bytes (to be) moved : 11027456

Tier Name        Size (bytes)    Space Before (bytes)   Space After (bytes)
tier4            268435456       210195456              221222912
tier3            268435456       268419072              268419072
tier2            268435456       268419072              268419072
tier1            268435456       250970112              239942656
```

Until a file placement policy has been refined through experience, the relocation component in particular may have unintended implications, such as unexpected file relocations due to `SELECT` statements that are written too inclusively. But adjusting a policy and enforcing it when a file system is busy with application I/O, can affect application performance adversely. To minimize the impact of file relocation on application performance, administrators can analyze multi-volume file systems with active file placement policies to determine which files would be relocated, how much data would be moved, and what the resulting occupancy level on each storage tier would be if the active policy were enforced. Line `[2]` in Dialog 4-5 illustrates the use of the `fsppadm analyze` command to determine the impact of enforcing a file system's active policy. Lines `[3]-[9]` (reformatted slightly from the actual report for readability) list the files that would be relocated if the policy were to be enforced, along with their destination placement classes and the names of the rules under which they would be enforced. Line `[10]` contains the total number of bytes that would be relocated, and lines `[11]-[15]` list the estimated total and allocated space on each of the file system's placement classes when the `fsppadm analyze` command is issued and as it would be if the active policy were enforced immediately.

Enforcing the active file placement policy

A VxFS multi-volume file system enforces the new file allocation part of its active file placement policy when files are created. Existing files are not relocated automatically, however. To relocate existing files, the active policy must be enforced by an authorized administrator using the `fsppadm enforce` command, as illustrated in Dialog 4-6.

Dialog 4-6 Enforcing a file system's active file placement policy

```
[1]   # fsppadm enforce -r /tmp/new/rep /vsa

[2]   Files moved : 3
[3]   Bytes moved : 11027456

[4]   Tier Name  Size (bytes)  Space Before (bytes)  Space After (bytes)
[5]   tier4      268435456     210195456             221222912
[6]   tier3      268435456     268419072             268419072
[7]   tier2      268435456     268419072             268419072
[8]   tier1      268435456     250970112             239937536
```

The `-r` switch in the `fsppadm enforce` command causes the detailed relocation report to be written to the file `/tmp/new/rep`. This report lists files that were relocated as a result of policy enforcement.

File placement policy enforcement is typically scheduled as a `cron` job to run at regular intervals (daily, for example) to keep actual file locations in reasonable accord with the administrative intent for the file system as expressed in its active file placement policy. To minimize the effect of file relocation on application performance, the command should be scheduled to run at times when application activity against the file system is low. The frequency with which it should be scheduled depends upon the type of activity a file system experiences over time. File systems in which events that can make relocation desirable are frequent should have policy enforcement scheduled more frequently than less dynamic file systems.

File placement reports

If a report is generated during file placement policy enforcement, a listing of relocated files, the placement classes to which they were relocated, and the reasons for relocation is available for inspection. Dialog 4-7 illustrates the content of the `fsppadm enforce` report (reformatted slightly for readability).

Dialog 4-7 Report produced by the fsppadm enforce command

```
[1]      # cat /tmp/new/rep

[2]      Sun Nov 20 22:09:38 2005                                          - Database_class1
[3]      /vsa/NODE0001/SQL00001/SQL1.DAT                              - Relocated to - tier1
[4]      Sun Nov 20 22:09:41 2005                                         - Database_class1A
[5]      /vsa/suzi/inst01/NODE0000/SQL00001/SQL1.DAT                  - Relocated to - tier1
[6]      Sun Nov 20 22:09:42 2005                                         - Database_class1A
[7]      /vsa/suzi/inst01/NODE0000/SQL00001/SQL11.DAT                 - Relocated to - tier1
```

The report in Dialog 4-7 indicates that VxFS relocated file `SQL1.DAT` in directory `/vsa/NODE0001/SQL00001/` to `tier1` in accordance with rule `Database_class1`, and so forth.

Placement policy enforcement is typically scheduled to run from an authorized account at regular intervals, but it can also be run by an authorized administrator whenever necessary to cause immediate relocation and deletion according to the rules of the active file placement policy. In addition to entire file systems, the command can be run against specific files or sets of files. Unscheduled policy enforcement against a specific set of files can be particularly useful after an administrative action such as renaming a group of files changes conditions in such a way that identifiable sets of files are no longer located in accordance with the active policy.

The fsppadm query command

The `fsppadm query` command can be used by an authorized administrator to determine the locations of some or all of the files in a multi-volume file system and the placement policy rules that resulted in those locations, as Dialog 4-8 illustrates.

Dialog 4-8 Querying a file system to determine file locations

```
        # fsppadm query /vsa/NODE0001 /vsa/suzi

[1]     Created  Current   Relocated
[2]     Class    Class     Class     Rule                                File
[3]     tier1    tier1     tier1     Database_class1
[4]                                                 /vsa/NODE0001/SQL00001/SQL1.DAT
[5]     tier1    tier1     tier1     Database_class1A
[6]                                       /vsa/suzi/inst01/NODE0000/SQL00001/SQL1.DAT
[7]     tier1    tier1     tier1     Database_class1A
[8]                                      /vsa/suzi/inst01/NODE0000/SQL00001/SQL11.DAT
[9]     tier4
[10]                                       /vsa/suzi/inst01/NODE0000/SQL00001/passwd
[11]    tier4
[12]                                       /vsa/suzi/inst01/NODE0000/SQL00001/my.DAT
[13]    tier4
[14]                                       /vsa/suzi/inst01/NODE0001/SQL00001/passwd
[15]    tier4
[16]                                      /vsa/suzi/inst01/NODE0001/SQL00001/SQL1.DAT
```

In this example, slightly reformatted for readability, the `fsppadm query` command only reports on files residing in directory `/suzi` (relative to the file system's mount point, `/vsa`). The columns in the report indicate the placement classes on which files were initially created, the classes on which they reside when the command was issued, the classes to which they would be relocated if the policy were to be enforced, and the rule that determined their current locations. Lines `[9]`-`[16]` in Dialog 4-8 pertain to files that have not been relocated (their current placement class, `tier4`, is the same as the placement class upon which they were created), and would not be relocated if the active policy were to be enforced.

The fsppadm list command

Either the name or the contents of a file system's active file placement policy can be ascertained at any time. Dialog 4-9 illustrates the `fsppadm list` and `fsppadm print` commands, which list the name and content of a file system's active file placement policy respectively.

Dialog 4-9 Listing and printing file placement policies

```
[1]     # fsppadm list /vsa
[2]     MOUNT_POINT     PLACEMENT_POLICY
[3]     /vsa            DBED_database_policy

[4]     # fsppadm print /vsa
[5]     <!DOCTYPE PLACEMENT_POLICY SYSTEM "/opt/VRTSfspro/config/placement.dtd">
[6]     <PLACEMENT_POLICY Version="5.0" Name="DBED_database_policy"  >

.            ......File placement policy rules......
```

Dialog 4-9 Listing and printing file placement policies *(continued)*

```
[7]     </PLACEMENT_POLICY>
```

Chapter 5

Advanced Dynamic Storage Tiering usage

This chapter includes the following topics:

Advanced DST features

The Dynamic Storage Tiering facility has several advanced features that increase its utility and applicability. This chapter describes these unique features and their implications on multi-volume file system and DST behavior.

The DST value proposition

The primary value proposition of the VxFS Dynamic Storage Tiering facility is a reduction in average online storage hardware cost without an offsetting increase in administrative complexity. The reduction in hardware cost stems from placing files on storage devices commensurate with their business value. Critical files are placed on highly available, high-performance (and therefore expensive) storage devices; less critical files are placed on lower cost devices with lesser performance and availability specifications. The more non-critical data an enterprise must keep online, the greater the fraction of its online storage that can be purchased and configured to keep cost low.

The conventional strategy for utilizing multi-tier storage is to create separate file systems for each type of storage in the data center, and to copy files from one to another as business needs change. For example, files containing database tables of transactions can be moved to lower cost storage devices as the transactions age, because aged transactions are backed up, and applications and users access them infrequently if at all.

But as a data center grows, the number of file systems and administrative operations required to implement strategies of this sort becomes unwieldy. Each change in the way data is stored and organized must be accompanied by corresponding changes in application configurations and operating procedures. The VxFS Dynamic Storage Tiering facility automates the relocation of files to the right types of storage devices without changing their logical locations in the file system name space. Because physically relocated files remain at the same logical locations, DST eliminates the need for changes in applications and operating procedures. To applications and users, DST file relocation is transparent.

Volume accessibility: a key assumption behind multi-tier storage

An important, but usually unvoiced, assumption behind multi-tier storage is that not all storage devices need be accessible all the time for a system to function. By storing data on less failure-tolerant devices, an enterprise implicitly acknowledges that that data is less critical to its operation than other data stored on more failure-tolerant devices. In the extreme, an enterprise that stores data on devices that are not failure-tolerant implicitly acknowledges that it can function satisfactorily without that data, at least for the period required to install replacement storage devices and restore it from a backup copy.

If a storage device in a conventional data center is unavailable, the file system on it cannot be mounted, and applications that use that file system cannot run. With a multi-volume file system, however, only the most critical data must be accessible for the file system to function; storage devices containing non-critical data need not be accessible.

Like any commercial file system, a VxFS multi-volume file system must be able to access all of its metadata in order to function. But a VxFS file system does not require access to any particular user files' data in order to mount and run. If certain files' data is inaccessible, for example, because the volume containing it has failed, applications that require those files cannot run, but otherwise the file system functions as usual.

The conclusion from this is that a multi-volume file system's metadata should be placed on the most failure-tolerant volumes in its volume set, while user files' data can be placed on volumes that are commensurate with the files' business value.

VxFS metadata placement

VxFS provides administrative control over metadata placement with a `dataonly` flag for each volume in a file system's volume set. If a volume's `dataonly` flag is set, VxFS only stores user file data on it, not metadata. Volumes whose `dataonly` flag is not set are eligible to store metadata as well as user data.

The first volume in a VxFS file system's volume set (added when the `vxvset make` command is executed) is necessarily eligible to store metadata, a property known as `metadataok`. By default, VxFS sets all other volumes' `dataonly` flags at file system creation time, or as volumes are added to a volume set. If necessary, an administrator can use the `metadataok` flag, illustrated in Dialog 5-1, to clear a volume's `dataonly` flag when adding it to a file system's volume set, making the volume eligible to store metadata as well as user data.

Dialog 5-1 Adding a metadata-eligible volume to a volume set

```
[1]     # fsvoladm add -f metadataok /vsb vsavola 100m
```

The command in Dialog 5-1 adds volume `vsavola` to the volume set for the file system mounted at `/vsb`, and clears its `dataonly` flag, making it eligible to hold metadata as well as user data.

A volume's eligibility to hold metadata can also be changed after it is already part of a mounted file system's volume set. Dialog 5-2 shows the `fsvoladm clearflags` and `fsvoladm setflags` commands that make volumes eligible and ineligible to hold metadata respectively.

Dialog 5-2 Changing a volume's metadata eligibility

```
[1]     # fsvoladm clearflags dataonly /vsb vsavolb
[2]     # fsvoladm setflags dataonly /vsb vsavolc
```

When an administrator sets the `dataonly` flag of a volume that contains metadata, as in Line `[2]` in Dialog 5-2, VxFS removes all metadata from the volume (provided that there is sufficient free space elsewhere in the file system's volume set to relocate the metadata).

VxFS dataonly volume configuration considerations

By designating volumes as `dataonly`, administrators prevent VxFS from storing any file system metadata on them. Only a file system's most failure-tolerant volumes should be made eligible to store metadata. A best practice for large file systems is to build a file system around a highly failure-tolerant volume sized to accommodate the expected amount of metadata, and not make that volume part of any placement class named in the file system's file placement policy. Volumes in placement classes that are named in the policy should be allowed to default to being `dataonly`, with their failure tolerance and performance properties dictated by the business requirements of the user data directed to them by the policy. If metadata space becomes dangerously full, the critical volume can be expanded as needed.

Similar considerations apply to multi-volume file systems that are created by upgrading single-volume file systems. Volume 0 (the file system's original volume) should be made failure-tolerant, for example by the addition of mirrors, sized appropriately, and not specified in any file placement policy rules. With appropriately specified rules, policy enforcement will relocate files away from volume 0, eventually resulting in it holding only metadata. If other volumes added to the volume set are allowed to default to being `dataonly`, the result will ultimately be the same as for a file system originally created as a multi-volume one.

Partially accessible file systems

A VxFS multi-volume file system can be mounted and accessed if some,or even all, of its `dataonly` volumes are inaccessible. The data in files stored on inaccessible volumes cannot be accessed, although pure metadata operations on them, such as ownership or permission changes, are still possible. But files whose data is located on available volumes are accessible, and VxFS can manipulate all file system metadata.

Allocation of new files can fail because no free space is available for metadata, even though there may be adequate space available on `dataonly` volumes. Administrators should therefore allocate adequate space on `metadataok` volumes to meet anticipated needs, and should monitor free space on them on an ongoing basis. VxVM facilities can expand the capacity of `metadataok` volumes without disrupting file system access.

The `dataonly` designation has meaning only at the file system level. It is a completely distinct property from file placement class. Because VxFS treats all volumes in a placement class as equivalent when making initial allocation and relocation decisions, all volumes in a given placement class should be of comparable failure tolerance and I/O performance, and should have the same `dataonly` or `metadataok` property.

Using VxFS dataonly volumes

The `dataonly` designation is independent of volume type. Volumes whose `dataonly` flags are set can store user file data regardless of how such data is defined. For example, a policy might specify that an enterprise's most critical data be stored on highly redundant `dataonly` volumes, and relocated to lower-cost, but still redundant `dataonly` volumes as it becomes inactive.

Other rules might specify that temporary (`*.tmp`) files be stored on a low-cost non-redundant `dataonly` volume and never relocated. Because a VxFS file system does not require `dataonly` volumes to be present in order to mount, failure of the `dataonly` volume containing `tmp` files would prevent access to them, but would not affect other uses of the file system.

Similarly, a policy rule might relocate inactive files to non-redundant volumes. As long as the files have been backed up, there is little apparent need to keep them on expensive storage devices. Failure of a volume containing inactive files would not impact applications at all unless the files were accessed, at which time they could be restored from backup copies. In the meantime, other data in the file system would be fully accessible by applications.

Inaccessible dataonly volumes and file placement policies

When enforcing file placement policies, VxFS treats inaccessible `dataonly` volumes as though they were present, but fully occupied. If, according to a policy rule, a new, extended, or relocated file should be allocated on an inaccessible `dataonly` volume, actual placement of that file depends on other conditions in the file system and policy as indicated in Table 5-1.

Table 5-1 File placement when target volume is inaccessible

Condition	Result
Space is available on another volume in the inaccessible volume's placement class	The file is placed on the alternate volume.
The rule also specifies a lower-priority placement class with adequate free space	The file is placed on a volume in the lower-priority placement class.
For file creation only, `Flags=any` is specified in the `<ON>` clause of the rule's `CREATE` statement	VxFS chooses the volume on which to allocate the new file.
None of the above conditions is the case	New file allocation fails with an `ENXIO` indication, making it possible for applications to distinguish between inaccessible volumes and placement classes that are actually fully occupied. Existing files are not relocated, but there is no error indication.

Discovering file and volume relationships

In a multi-volume file system, knowing which volume holds a file can be an important piece of information for an administrator. For example, application performance problems can sometimes be diagnosed by noticing that two heavily accessed files occupy the same volume, and relocating one of them to an alternate. Failure of applications to initialize can often be traced to files that are inaccessible, for example, because they reside on inaccessible `dataonly` volumes.

The fsmap command

VxFS administrators can either use facilities of the Storage Foundation Management Server or run the `fsmap` command on the host that has a file system mounted to discover the volumes on which files are located, as Dialog 5-3 illustrates.

Dialog 5-3 Determining which volumes contain a given file

```
[1]      # fsmap -q /vsb/suzi/inst01/NODE0000/SQL00001/*
[2]            vsbvola          Data      /vsb/suzi/inst01/NODE0000/SQL00001/SQL11.DAT
```

Dialog 5-3 Determining which volumes contain a given file *(continued)*

```
[3]          vsbvola         Data     /vsb/suzi/inst01/NODE0000/SQL00001/SQL2.DAT
[4]          vsbvolb         Data     /vsb/suzi/inst01/NODE0000/SQL00001/SQL4.IN1
```

The `fsmap` command in line `[1]` of Dialog 5-3 requests the locations of all files in directory `/vsb/suzi/inst01/NODE0000/SQL00001/`. The response indicates that the database data files (`.DAT`) are on `vsbvola` and the initialization file (`.INI`) is on `vsbvolb`.

The fsvmap command

Likewise, it can be equally important to know which files occupy a particular volume, for example, to assess the impact of administrative operations such as volume expansion or relocation on production applications. Administrators of VxFS file systems can use the `fsvmap` command to determine which files and directories reside on a particular volume. Dialog 5-4 illustrates the use of the `fsvmap` command to list the files in the multi-volume file system mounted at `/vsb` that occupy space on volume `vsbvola`.

Dialog 5-4 Determining which files are located on a given volume

```
[1]      # fsvmap /vsb vsbvola
[2]      /
[3]      /lost+found/
[4]      /dir/dir1/dir2/
[5]      /dir/dir1/dir2/A.DAT
[6]      /dir/dir1/dir2/B.UXT
[7]      /dir/dir1/dir2/SQLTAG.NAM
[8]      /dir/dir1/dir2/SQL00057.INX
[9]      /dir/dir1/dir3/SQL00057.INX
[10]     /dir/dir1/dir3/SQLTAG.NAM
[11]     /dir/dir1/dir3/nomove
[12]     /dir/dir1/dir4/SQLBP.2
[13]     /dir/dir1/dir4/SQLBP.1
[14]     /dir/dir1/dir4/core
[15]     /suzi/inst01/NODE0000/SQL00001/.
[16]     /suzi/inst01/NODE0000/SQL00001/SQLa.INX
[17]     /suzi/inst01/NODE0000/SQL00001/SQL4.INI
[18]     /suzi/inst01/NODE0000/SQL00001/SQL11.DAT
[19]     /suzi/inst01/NODE0000/SQL00001/SQL2.DAT
[20]     /suzi/inst01/NODE0000/SQL00001/SQL6.INX
[21]     /suzi/inst01/NODE0000/SQL00001/SQL6.LB
[22]     /suzi/inst01/NODE0000/SQL00001/SQL7.LB
[23]     /suzi/inst01/NODE0000/SQL00001/SQL8.LB
```

The `fsvmap` command can be used to evaluate the effectiveness or appropriateness of the active file placement policy. It is also useful for determining which files would be affected by a volume size or layout change, or alternatively, which files are inaccessible when a `dataonly` volume fails. In the latter case, the output of the `fsvmap` command can be edited for use as input to NetBackup's `BPRESTORE` program to restore files that are lost as a result of a `dataonly` volume failure onto alternative or replacement volumes in the file system's volume set. A useful byproduct of this capability is that by comparing files' `atime`s in the file system metadata (still accessible, because they are not stored on the failed `dataonly` volume) with the `atime`s recorded in the backup image, administrators can determine which files restored from the backup are actually out-of-date and which have been restored to their states at the time of the volume failure.

Placing data restored from file-based backups

From a placement policy standpoint, restoring a file is equivalent to creating a new file. VxFS places files restored from backups according to the file system's active placement policy rules expressed in `CREATE` statements. Restored files are not relocated until the policy is next enforced. This may have unexpected effects under certain circumstances. For example, if a file system containing a large number of inactive files that have been relocated to second-tier storage experiences a second-tier volume failure, files from the failed volume are restored to the placement classes on which they were originally allocated, rather than those to which they had been relocated due to inactivity. Restoring many such files might result in top tier storage filling up with data that is actually inactive. Allocation may even fail for some restored files if the `Flags=any` does not appear in the `<ON>` clauses of `CREATE` statements. Moreover, there may be significant time and bandwidth consumption during the first few policy enforcements following the restore as inactive restored files are relocated to second-tier volumes.

Distributed file allocation

Distributing data across two or more storage devices is a well-known technique for improving I/O performance. The sequential access typical of streaming applications tends to improve because large I/O requests are broken up into parts which transfer data to or from multiple devices concurrently. Random access, typical of transaction processing applications, also tends to improve because multiple I/O requests to a data set tend to execute concurrently using separate physical resources.

Striping is well-known as a technique for distributing virtual volume or LUN block addresses across physical storage devices. When a volume's block addresses are striped across multiple disks or LUNs, the larger files stored on it tend to have their data striped across multiple devices.

Striping at the volume level distributes data for all files in the file system on the volume. But it can also be useful to select the files whose blocks should be distributed, rather than indiscriminately striping all files that occupy a volume. For example, the predominant processing mode for some database tables is sequential. The files that contain these tables would benefit from concurrent execution of large read requests, whereas other files in the same file system might be more suitably placed on individual volumes.

The Dynamic Storage Tiering facility can distribute the storage space occupied by selected files across the volumes in a placement class. A `<DESTINATION>` specification within the `<ON>` clause of a `CREATE` statement or within the `<TO>` clause of a `RELOCATE` statement may optionally specify a `<BALANCE_SIZE>` sub-element. If a `<BALANCE_SIZE>` is specified, and if the destination placement class contains more than one volume, VxFS allocates storage for files on the placement class (both at creation time and when the file is extended) in extents of `<BALANCE_SIZE>` size, distributing them randomly across all of the volumes in the placement class.

Fragment 5-1 illustrates the use of `<BALANCE_SIZE>` to distribute the space allocated to selected files across the volumes in a placement class.

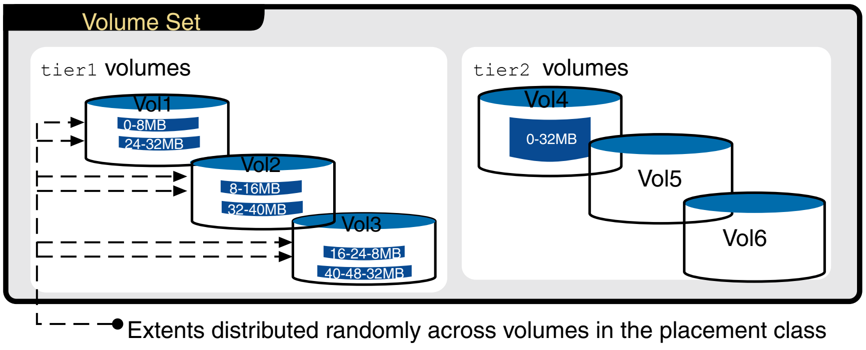
Fragment 5-1 Distributed space allocation

```
<RULE Name=DistributedRule>
  <SELECT><PATTERN>*.db</PATTERN></SELECT>
  <CREATE><ON>
    <DESTINATION>
      <CLASS>tier1</CLASS>
      <BALANCE_SIZE Units="MB">8</BALANCE_SIZE>
    </DESTINATION>
    <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
  </ON></CREATE>
</RULE>
```

In this example, files whose names follow the pattern `*.db` are allocated on `tier1` volumes in extents of 8 megabytes distributed randomly across the `tier1` volumes. If a selected file cannot be allocated on `tier1` volumes because they are full, it is allocated on a `tier2` volume, but the space allocated to it is not distributed across multiple `tier2` volumes.

Figure 5-1 illustrates two potential allocations of 32-megabyte files on `tier1` and `tier2` volumes according to the rule in Fragment 5-1.

Figure 5-1 VxFS Distributed space allocation



As Figure 5-1 illustrates, a 32 megabyte file would be allocated in four 8-megabyte extents distributed randomly across the three `tier1` volumes. Because the `<DESTINATION>` specification for `tier2` in Fragment 5-1 does not contain a `<BALANCE_SIZE>` sub-element, a 32-megabyte file forced to `tier2` because `tier1` volumes were fully occupied would be allocated on a single volume.

VxFS-directed file placement

When a multi-volume file system has an active file placement policy, VxFS attempts to allocate space for new files in the placement classes specified in the `<ON>` clause of the `CREATE` statement of the applicable rule for each file. If allocation fails because no destination has sufficient free space, file creation fails with an `ENOSPC` indication, even though there may be adequate space in other placement classes. In some cases, this may be the desired behavior. In others, it may be preferable to permit VxFS to use any free space available to allocate a new file as a last resort, rather than fail allocation because no space is available in any of the designated destination placement classes.

Including the `Flags=any` attribute in the `<ON>` clause of a `CREATE` statement causes VxFS to use any available free space to allocate or extend selected files if no designated destination placement class has adequate free space. The rule in lines `[1]-[7]` of Fragment 5-2 illustrates the use of the `any` attribute to cause allocation of files whose names follow the pattern `*.db` to succeed even if no `tier1` or `tier2` space is available.

Fragment 5-2 Example of VxFS-determined file placement

```
[1]        <RULE Name=AllocateAnywhereRule>
[2]         <SELECT><PATTERN>*.db</PATTERN></SELECT>
[3]         <CREATE><ON Flags=any>
[4]             <DESTINATION><CLASS>tier1</CLASS></DESTINATION>
[5]             <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
[6]           </ON> </CREATE>
[7]        </RULE>

           <RULE Name=RestrictedAllocationRule>
[8]         <SELECT><PATTERN>*</PATTERN></SELECT>
[9]         <CREATE><ON>
[10]            <DESTINATION><CLASS>tier1</CLASS></DESTINATION>
[12]            <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
[13]        </ON></CREATE>
[14]      </RULE>
```

By the first rule, files whose names follow the pattern are allocated on `tier1` volumes if possible, `tier2` volumes if no `tier1` space is available, and at locations elsewhere in the file system's volume set chosen by VxFS if neither `tier1` nor `tier2` space is available. Allocation of `*.db` files only fails if there is no free space anywhere in the file system's volume set. The rule in lines `[8]-[14]`, does not include the `Flags=any` attribute, so allocation of files whose names do not follow the pattern `*.db` fails with an `ENOSPC` indication if sufficient space is not available on `tier1` or `tier2` volumes.

Some `dataonly` volumes may be unavailable to a mounted file system, for example, because they have failed. When a rule's `CREATE` statement indicates that space for a new or extended file should be allocated on an inaccessible `dataonly` volume, VxFS treats the file as though the volume were accessible but fully occupied. The file is allocated on another volume in the same placement class if possible, in a lower-priority destination placement class if not, and allocation failure with `ENXIO` unless the `CREATE` statement's `<ON>` clause contains the `Flags=any` attribute. VxFS returns the `ENXIO` failure code to permit applications to distinguish between failed volumes and fully occupied placement classes. In the latter case, `ENOSPC` is returned.

The effect of statement ordering on file relocation

The order in which `RELOCATE` and `DELETE` statements appear in a file placement policy rule can affect the rule's behavior. A VxFS policy rule may contain one or more `RELOCATE` statements, each of which may include any or all of the optional elements listed in Table 5-2.

Table 5-2 Elements specified in a RELOCATE statement

Element	Description
Source placement classes	An unordered list of placement classes from which qualifying files selected by the rule are relocated
Destination placement classes	A priority-ordered list of placement classes to which qualifying files selected by the rule are relocated
Relocation qualifiers	Up to five relocation qualifiers which, if met, qualify files selected by the rule to be relocated

If a rule does not contain any `RELOCATE` statements, files to which it applies are never relocated, even if other rules occurring later in the policy would result in their relocation. Fragment 5-3 illustrates this scenario.

Serial application of policy rules

Fragment 5-3 illustrates the serial application of the rules in a file placement policy.

Fragment 5-3 Example of a VxFS selective relocation policy

```
[1]      <RULE Name=DatafileRule>
[2]        <SELECT><PATTERN>*.dat</PATTERN></SELECT>
[3]        <CREATE><ON>
[4]          <DESTINATION><CLASS>tier1</CLASS></DESTINATION>
[5]          <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
[6]        </ON></CREATE>
[7]      </RULE>

[8]      <RULE Name=CatchallRule>
[9]        <SELECT><PATTERN>*</PATTERN></SELECT>
[10]       <CREATE><ON><DESTINATION><CLASS>tier2</CLASS></DESTINATION></ON></CREATE>
[11]       <RELOCATE>
[12]         <TO><DESTINATION><CLASS>tier3</CLASS></DESTINATION></TO>
[13]         <WHEN>......relocation qualifiers......</WHEN>
[14]       </RELOCATE>
[15]     </RULE>
```

In the first rule of this example (lines `[1]-[7]`), files with a name extension of `dat` following the period are allocated on `tier1` volumes if possible, and on `tier2` volumes if no `tier1` space is available. This rule contains no relocation clause, so `dat` files are never relocated.

The catchall rule (lines `[8]-[15]`) causes all other files to be allocated on `tier2` volumes until those volumes are fully occupied, after which allocation fails. This rule specifies relocation of selected files to `tier3` volumes if the qualifications specified in the `<WHEN>` clause are met. Because `dat` files are selected by the first rule, the second rule never applies to them. Since the first rule contains no relocate statements, `dat` files are never relocated.

Ordering of RELOCATE and DELETE statements

VxFS qualifies each file to which a rule applies against the rule's `RELOCATE` and `DELETE` statements in order of occurrence, stopping when an applicable statement is encountered. Thus, the order of occurrence of `RELOCATE` and `DELETE` statements can affect their behavior. For example, Fragment 5-4 illustrates a rule for which the order of `RELOCATE` statements might lead to unintended results.

Fragment 5-4 Example of an ineffective file relocation policy statement

```
[1]       <RULE Name=LargeFilesRule>
[2]         <SELECT><PATTERN>*</PATTERN></SELECT>
[3]         <CREATE><ON><DESTINATION><CLASS>tier1</CLASS></DESTINATION></ON></CREATE>
[4]         <RELOCATE>
[5]           <FROM><SOURCE><CLASS>tier1</CLASS></SOURCE></FROM>
[6]           <TO><DESTINATION><CLASS>tier2</CLASS></DESTINATION></TO>
[7]           <WHEN><SIZE Units="MB"><MIN Flags="gt">50</MIN></SIZE></WHEN>
[8]         </RELOCATE>

[9]         <RELOCATE>
[10]          <FROM><SOURCE><CLASS>tier1</CLASS></SOURCE></FROM>
[11]          <TO><DESTINATION><CLASS>tier3</CLASS></DESTINATION></TO>
[12]          <WHEN><SIZE Units="MB"><MIN Flags="gt">100</MIN></SIZE></WHEN>
[13]        </RELOCATE>
[14]      </RULE>
```

The first `RELOCATE` statement in lines `[4]-[8]` of Fragment 5-4 relocates files on `tier1` volumes that are larger than 50 megabytes to `tier2` volumes. The second `RELOCATE` statement in lines `[9]-[13]` relocates files on `tier1` volumes that are larger than 100 megabytes to `tier3` volumes. But since a file that is larger than 100 megabytes is necessarily larger than 50 megabytes, any file that would qualify for the second RELOCATE statement also qualifies for the first. Therefore, the second rule never applies to any file.

In Fragment 5-4, the intent is clearly to create all files on `tier1` volumes, and relocate them from `tier1` to `tier2` volumes when they grow larger than 50 megabytes, or from `tier1` to `tier3` volumes when they grow larger than 100 megabytes. The desired intent can be achieved by reversing the order of the two `RELOCATE` statements.

Placement policy enforcement relocates files selected by a rule that reside in a placement class specified in the `<FROM>` clause of a `RELOCATE` statement and meet the qualifications specified in the `<WHEN>` clause. If a relocate statement does not include a <FROM> clause, qualifying files residing anywhere in the file system's volume set are eligible for relocation.

Files are relocated the highest-priority placement class listed in the <TO> clause on which space is available. Relocation may not occur if free space on destination volumes is consumed by other applications between the time a file is selected for relocation and the time that an enforcement thread actually attempts to relocate it.

Overlapping placement classes

The same placement class may appear in both <FROM> and <TO> clauses of a RELOCATE statement. Fragment 5-5 illustrates a RELOCATE statement in which tier2 and tier3 appear as both sources and destinations for file relocation.

Fragment 5-5 Overlapping storage classes in a RELOCATE statement

```
<RELOCATE>
  <FROM>
    <SOURCE><CLASS>tier1</CLASS></SOURCE>
    <SOURCE><CLASS>tier2</CLASS></SOURCE>
    <SOURCE><CLASS>tier3</CLASS></SOURCE>
  </FROM>
  <TO>
    <DESTINATION><CLASS>tier4</CLASS></DESTINATION>
    <DESTINATION><CLASS>tier3</CLASS></DESTINATION>
    <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
  </TO>
  <WHEN><ACCAGE Units="Days"><MIN Flags="gt">30</MIN></ACCAGE></WHEN>
</RELOCATE>
```

In the <FROM> clause in Fragment 5-5, the order in which the placement classes appear is immaterial. VxFS relocates files in any of the listed placement classes if they meet the qualification in the statement's <WHEN> clause. In the <TO> clause, however, the placement class list is priority-ordered, with the most desirable destination listed first. Policy enforcement does not relocate a file from a placement class that is higher in the destination placement class list than the highest priority class with available space. Referring to the RELOCATE statement in Fragment 5-5, for example, if tier4 volumes were fully occupied, VxFS would not relocate files from tier3 volumes to tier2 volumes because their current location (tier3) has a higher priority than placement class tier2.

Unconditional file deletion

Fragment 5-6 illustrates a policy rule that unconditionally deletes `.mp3` files belonging to user `user1` each time the rule is enforced.

Fragment 5-6 VxFS unconditional deletion policy rule

```
<RULE Name=NoMP3Rule>
  <SELECT><PATTERN>*.mp3</PATTERN><USER>user1</USER></SELECT>
  <DELETE></DELETE>
</RULE>
```

This rule deletes all `mp3` files owned by `user1`, regardless of their locations, whenever VxFS enforces the active policy.

Additional policy rule ordering considerations

Fragment 5-7 contains a series of policy rules that illustrate certain additional subtleties of policy rule ordering.

Fragment 5-7 The effect of rule order in a file placement policy

```
[1]         <RULE Name=DatafilesRule>
[2]          <SELECT><PATTERN>*.dat</PATTERN></SELECT>
[3]          <CREATE><ON><DESTINATION><CLASS>tier2</CLASS></DESTINATION></ON></CREATE>
[4]         </RULE>

[5]         <RULE Name=User1FilesRule>
[6]          <SELECT><USER>user1</USER></SELECT>
[7]          <CREATE><ON><DESTINATION><CLASS>tier1</CLASS></DESTINATION></ON></CREATE>
[8]         </RULE>

[9]         <RULE Name=User1FilesRule>
[10]         <SELECT><USER>user2</USER></SELECT>
[11]         <CREATE><ON><DESTINATION><CLASS>tier2</CLASS></DESTINATION></ON></CREATE>
[12]        </RULE>

[13]        <RULE Name=TempFilesRule>
[14]         <SELECT><PATTERN>*.tmp</PATTERN></SELECT>
[15]         <CREATE><ON><DESTINATION><CLASS>tier3</CLASS></DESTINATION></ON></CREATE>
[16]        </RULE>
```

The rule in lines `[1]-[4]` of Fragment 5-7 causes files whose names follow the pattern `*.dat` to be created on `tier2` volumes. The rules in lines `[5]-[12]` specify that all files owned by `user1` and `user2` are to be created on `tier1` and `tier2` volumes respectively. But because files are created according to the first applicable rule in the policy, any `dat` files created by `user1` are created on `tier2` volumes, which may not have been the administrative intention.

The rule in lines `[13]-[16]` causes files whose names follow the pattern `*.tmp` to be created on `tier3` volumes. But since the second and third rules apply to all files created by `user1` and `user2`, including `tmp` files, VxFS places `tmp` files that belong to these users on `tier1` and `tier2` volumes respectively. Again, this may not be the intention of the policy.

Placing the second and third rules ahead of the first would result in `user1` and `user2`'s `dat` files being created on `tier1` and `tier2` volumes respectively. Similarly, placing the second and third rules after the fourth rule would cause `user1` and `user2`'s `tmp` files to be created on `tier3` volumes. In this case, however, `user1`'s `dat` files would be created on `tier2` volumes.

There is no right or wrong answer in this example. Its point is to emphasize that administrators must understand their storage management goals clearly and express them precisely when they define file placement policies. Combinations of the `fsmap`, `fsvmap`, and `fsppadm query` utilities can help determine both the initial state of a file system and the effect that enforcement of the active policy would have.

Enterprise-wide file placement policies

While the VxFS Dynamic Storage Tiering facility can be administered on the server whose file systems it controls, a very significant additional benefit for large data centers and enterprises is promotion of consistent online storage management across all of a data center's (and indeed, an enterprise's) computer systems. In data centers that have the Storage Foundation Management Server installed, a storage architect can create a master file placement policy, store it in the SFMS database, and propagate it from the management server to any number of file systems across the enterprise provided that they use the same `vxfs.placement class` volume tag values. A master policy is syntactically identical to a policy created for a single file system. It becomes a master by virtue of being assigned to multiple file systems by the Storage Foundation Management Server.

Standardizing volume tags

File placement policies specify the allocation and relocation of files between different placement classes. Volumes' placement classes are defined by the values of `vxfs.placement_class` volume tags assigned to them. For a policy to be applicable to a file system, the volumes in its volume set must have tags that appear in the policy. Unlike volume names, volume tags need not be unique. The recommended practice is for a data center to define standard volume tags that correspond to specific sets of performance and availability properties, and assign tags to volumes based on the volumes' properties. Because file placement policies refer only to placement classes, and not to specific volume names, a policy can be applied to any file system whose volume set is assigned tags that appear in the policy.

Because DST treats the conditions all volumes in placement class `x` fully occupied and no volumes of placement class `x` in volume set, identically, it is not necessary for all placement classes named in a policy to be represented in the volume sets of all file systems to which the policy is assigned.

Policies for file systems with different placement class sets

Fragment 5-8, Fragment 5-9, and the diagrams that follow use a file placement policy that names four placement classes to illustrate this point.

Fragment 5-8 Four-class file placement policy: Database_Rule

```
<?xml version="1.0"?>
<!DOCTYPE FILE_PLACEMENT_POLICY SYSTEM "placement.dtd">
<FILE_PLACEMENT_POLICY Name="Universal_Policy" Version="5.0">
<RULE Name=Database_Rule>
  <SELECT>
    <PATTERN>*.db</PATTERN>
    <PATTERN>*.ora</PATTERN>
    <PATTERN>*.dblog</PATTERN>
    <PATTERN>*.oralog</PATTERN>
  </SELECT>
  <CREATE><ON Flags=Any>
    <DESTINATION><CLASS>database</CLASS></DESTINATION>
    <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
  </ON></CREATE>
  <RELOCATE>
    <TO><DESTINATION><CLASS>tier3</CLASS></DESTINATION></TO>
    <WHEN><ACCAGE Units="days"><MIN Flags="gt">45</MIN></ACCAGE></WHEN>
  </RELOCATE>
  <RELOCATE>
    <TO><DESTINATION><CLASS>database</CLASS></DESTINATION></TO>
    <WHEN><ACCAGE Units="days"><MAX Flags="lt">3</MAX></ACCAGE></WHEN>
  </RELOCATE>
</RULE>
```

Fragment 5-8 shows the policy rule that applies to certain database files, which are selected by naming pattern. Selected files are created on `database` volumes if space permits, on `tier2` volumes if not, and elsewhere within the file system's volume set if both `database` and `tier2` volumes are fully occupied. Selected files are relocated to `tier3` volumes if they are not accessed for 45 days, and back to `database` volumes if they reside in other placement classes and have been accessed within three days of the time at which the policy is enforced.

Fragment 5-9 shows the catchall policy rule for all other files in the file system. Files are created on `tier1` volumes if space permits, on `tier2` volumes if not, on `tier3` volumes if neither `tier1` volumes nor `tier2` volumes have sufficient free space. If files cannot be allocated on `tier1`, `tier2`, or `tier3` volumes, allocation fails with an `ENOSPC` indication.

Fragment 5-9 Four-class file placement policy rule: Ordinary_Files_Rule

```
  <RULE Name=Ordinary_Files_Rule>
   <SELECT><PATTERN>*</PATTERN></SELECT>
   <CREATE><ON>
       <DESTINATION><CLASS>tier1</CLASS></DESTINATION>
       <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
       <DESTINATION><CLASS>tier3</CLASS></DESTINATION>
    </ON></CREATE>

    <RELOCATE>
     <TO>
       <DESTINATION><CLASS>tier3</CLASS></DESTINATION>
       <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
     </TO>
     <WHEN><ACCAGE Units="days"><MIN Flags="gt">30</MIN></ACCAGE>
     </WHEN>
    </RELOCATE>

    <RELOCATE>
     <TO><DESTINATION><CLASS>tier1</CLASS></DESTINATION></TO>
     <WHEN><IOTEMP>
       <MIN Flags="gt">6</MIN>
       <PERIOD>3</PERIOD>
     </IOTEMP></WHEN>
   </RELOCATE>
 </RULE>
</FILE_PLACEMENT_POLICY>
```

According to the rule in Fragment 5-9, selected files that are not accessed for more than 30 days are relocated to `tier3` volumes if possible, and to `tier2` volumes if not. Files whose three-day average I/O temperatures rise above 6 are relocated to `tier1` volumes.

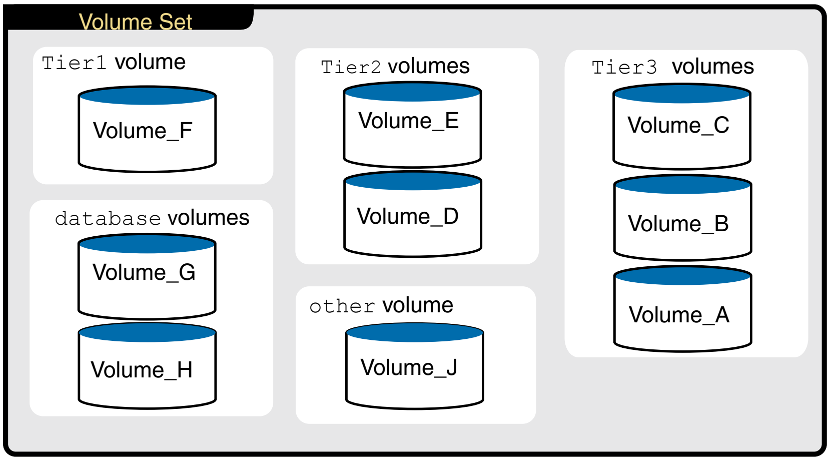
This policy is suitable for enterprise-class servers with three distinct tiers of volumes for general use (for example, based on mirrored LUNs presented by an enterprise disk array, mirrored mid-range LUNs, and low-cost SATA-based RAID-5 LUNs, respectively) as well as a fourth class of volumes specifically for database use. Other types of systems might not have volumes in all four placement classes. For example, systems that do not routinely run database applications might not have dedicated database volumes. Similarly, smaller systems might be configured with only two classes of storage for general use.

The Dynamic Storage Tiering facility only relocates files within placement classes specified in a file system's active policy. Placement classes in a file system's volume set that are not named in its active policy may be used for initial allocation if the keyword `any` is specified in one or more `CREATE` statement `<ON>` clauses, but VxFS does not relocate files to such volumes. (It may relocate files from them if the active policy contains `RELOCATE` statements without `<FROM>` clauses.) The examples that follow illustrate how the policy shown in Fragment 5-8 and Fragment 5-9 works with multi-volume file systems whose volume sets have different placement class configurations.

Applying a policy to file systems with different placement class sets

Figure 5-2 begins the discussion by illustrating a volume set that contains volumes in all four of the placement classes named in the policy.

Figure 5-2 Multi-volume file system with four placement classes

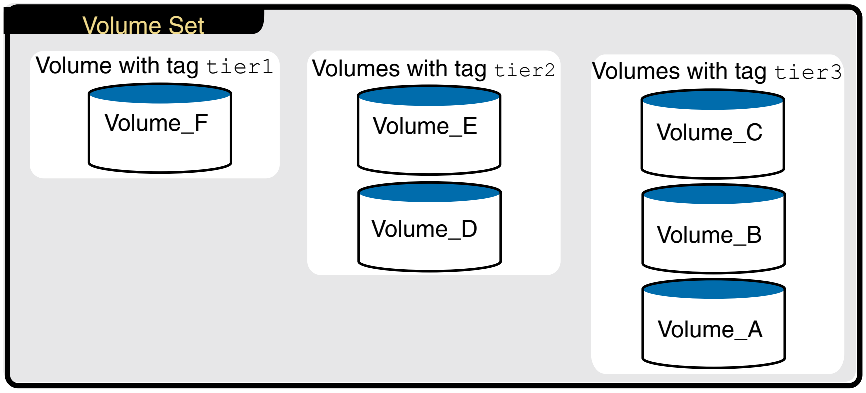


The volume set illustrated in Figure 5-2 contains all placement classes listed in the policy, plus an additional `other` class. No files are relocated to the `other` volume. Database files may be created on it (because the `any` keyword is specified in the `<ON>` clause for database files), but other files cannot, because the policy rule in Fragment 5-9 completely specifies permissible locations for newly-created files, and the `other` placement class is not among the specifications.

File systems with three placement classes

The point of presenting this policy is to illustrate that it can be applied to file systems whose volume sets do not contain volumes in all placement classes mentioned in the policy. For file creation and relocation purposes, DST treats placement classes that are not present in a file system's volume set as though their volumes were completely occupied. To illustrate this principle, Figure 5-3 shows a file system whose volume set contains three placement classes, with the same names as the three general-purpose classes in Figure 5-2, but no volumes in the `database` placement class.

Figure 5-3 Multi-volume file system with three placement classes



The placement class configuration shown in Figure 5-3 might be an installation standard for file systems that do not routinely host databases. If database files were to be allocated in a file system with this volume set configuration, VxFS would place them on `tier2` volumes according to the database policy rule (Fragment 5-8, line `[13]`). VxFS treats the (absent) `database` class as though all of its volumes were full. If `tier2` volumes fill, VxFS would allocate database files on other storage tiers because the `<ON>` clause of the rule's `CREATE` statement has the `any` attribute.

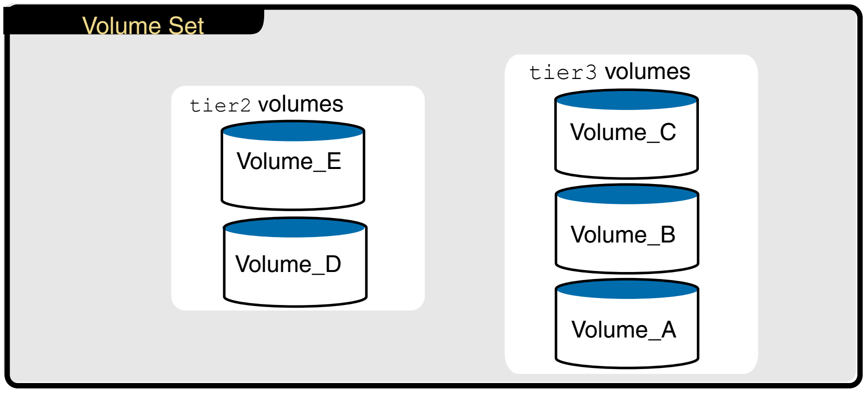
With this storage configuration, database files would be relocated to `tier3` volumes if they were not accessed for 45 days, but because there are no `database` volumes, they would never be relocated back from `tier3` volumes. This (possibly unintended) side effect could be remedied by specifying a secondary destination in the `RELOCATE` statement (following line `[20]` in Fragment 5-8).

The `Ordinary_Files_Rule` of this policy would cause VxFS to create non-database files on the `tier1` volume and relocate them to the `tier3` and `tier2` volumes based on inactivity, and back to the `tier1` volume based on increased average I/O temperature.

File systems with two placement classes

Carrying the concept further, the policy in Fragment 5-8 and Fragment 5-9 can also be applied to file systems whose volume sets contain only `tier2` and `tier3` placement classes, as illustrated in Figure 5-4.

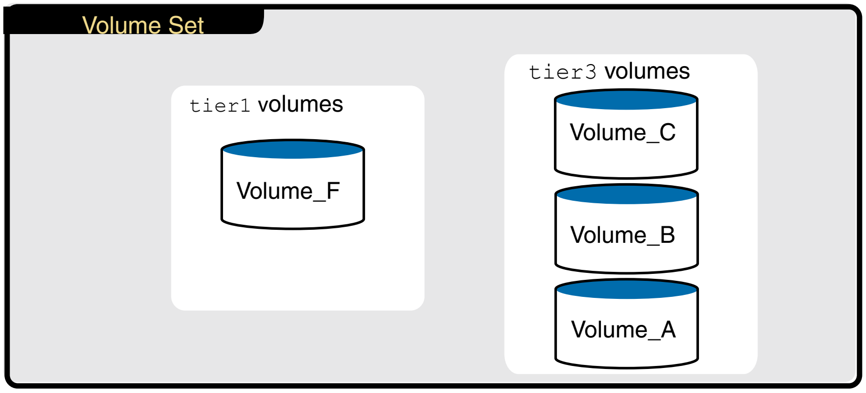
Figure 5-4 Multi-volume file system with two placement classes (alternative 1)



File systems on volume sets like that shown in Figure 5-4 might be configured on smaller systems that are not connected to enterprise-class (`tier1`) disk arrays. On such systems, database files would be created on `tier2` volumes because there are no `database` volumes, and relocated to `tier3` volumes if they went 45 days without being accessed. Once relocated to `tier3` volumes, database files would not be relocated again. This possibly unintended behavior could be altered by specifying a secondary destination in the `<TO>` clause of the relocation statement in lines `[19]-[22]` of Fragment 5-8. Non-database files would be allocated on `tier2` or `tier3` volumes, because there are no `tier1` volumes. Inactive files would be relocated from `tier2` volumes to `tier3` ones, and files with increased average I/O temperatures would be relocated from `tier3` volumes to `tier2`.

Figure 5-5 illustrates another file two-class volume set configuration to which this policy can be applied.

Figure 5-5 Multi-volume file system with two placement classes (alternative 2)

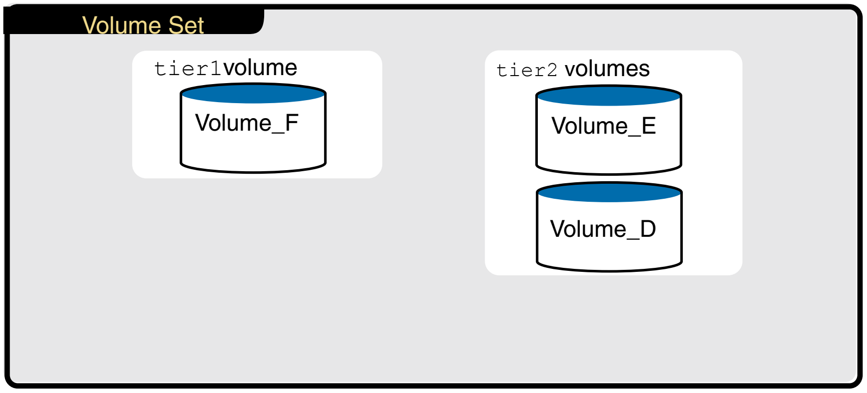


The volume set shown in Figure 5-5 contains only `tier1` and `tier3` volumes. It might be representative of a file system on a small system connected to a limited amount of enterprise disk array storage (`tier1`), and a larger complement of lower-cost (`tier3`) storage. Assigning the policy in Fragment 5-8 and Fragment 5-9 to a file system with this volume set configuration would result in VxFS choosing the locations for any new database files, because there are no `database` or `tier2` volumes. Database files could be relocated to `tier3` volumes, but relocation would be one-way, because there are no `database` volumes. With this configuration, adding `tier2` as a secondary destination in the `RELOCATE` statement in lines `[19]-[22]` of Fragment 5-8 would have no effect, because there are no `tier2` volumes in the file system's volume set.

VxFS would allocate non-database files on the `tier1` volume or one of the `tier3` volumes in this volume set, depending on space availability. Access age-based relocation would occur from `tier1` to `tier3` volumes, and I/O load-based relocation would occur from `tier3` volumes to `tier1` ones.

Figure 5-6 illustrates the third possible volume set configuration using only the general-purpose placement classes, one that contains only `tier1` and `tier2` volumes. This configuration, too, is consistent with the policy in and Fragment 5-9.

Figure 5-6 Multi-volume file system with two placement classes (alternative 3)



Assigning the policy in Fragment 5-8 and Fragment 5-9 to a file system with a placement class configuration like the one illustrated in Figure 5-6 would result in database files being allocated on `tier2` volumes if space were available, and on the `tier1` volume if not, because VxFS would choose locations for database files if no `tier2` space were available. Database files would never be relocated, because there are no `tier3` volumes to serve as destinations. VxFS would allocate non-database files on `tier1` or `tier2` volumes, depending on space availability. Inactive files would be relocated from `tier1` volumes to `tier2` volumes, and files with increased average I/O temperatures would be relocated from `tier2` volumes to `tier1` volumes, when the policy was enforced.

The policy in Fragment 5-8 and Fragment 5-9 can also be used with a file system whose volume set contains a single class of general storage volumes and the `database` class. For example, if `tier1` and `database` volumes were present on a system, both files and databases could be allocated, but no relocation would occur.

Summary: considerations for enterprise-wide file placement policies

Collectively, these examples illustrate the importance of coordinating the definition of a file placement policy for use with multiple file systems and placement class configurations for those file systems' volume sets, so that the desired behavior results wherever the policy is assigned. If enterprise-wide placement policies are coordinated with volume set configurations, the result is data center or even enterprise standards for file placement. This outcome supports the larger multi-tier storage goal of simplifying online storage management across a data center by introducing storage configuration consistency across large numbers of file systems.

Chapter 6

Storage Foundation built-in file placement policies

This chapter includes the following topics:

The built-in placement policy subset

The Veritas Enterprise Administrator (VEA) graphical management console includes a set of wizards that administrators can use to create and assign any of the four types of file placement policies listed in Table 6-1.

Table 6-1 VxFS built-in file placement policy types

File placement policy type	Description
Access age-based	This policy uses file access age (time since last access) to qualify files for downward relocation within the storage hierarchy. Files with larger access ages are relocated to lower tiers. This policy uses average I/O temperature to qualify files for upward location within the hierarchy. Files with higher average I/O temperatures are relocated to higher storage tiers.
Update age-based	This policy uses files' modification age (time since last modification) to qualify files for both downward and upward relocation within the storage hierarchy. Files with larger modification ages are relocated to lower storage tiers; files with smaller modification ages are relocated upward.

Table 6-1 VxFS built-in file placement policy types *(continued)*

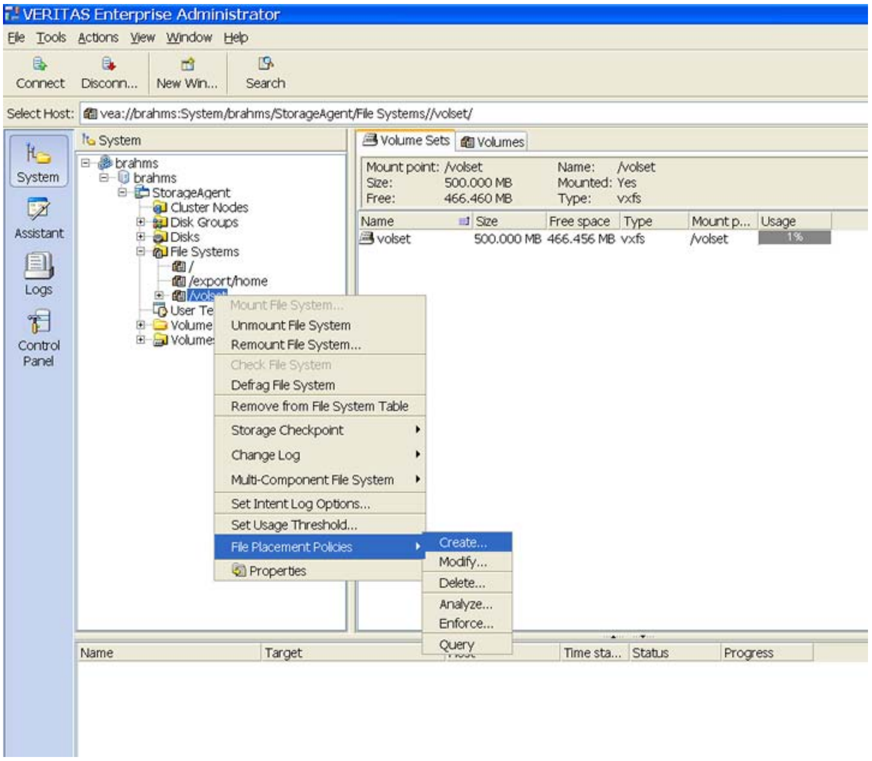
File placement policy type	Description
I/O activity-based	This policy uses files' average I/O temperatures to qualify them for relocation both downward and upward within the storage hierarchy. Files with lower average I/O temperatures are relocated to lower tiers; files with higher average I/O temperatures are relocated upward.
Access-age-based with preferred files	This policy is identical to the access age-based policy, except that a specified set of files is placed in a dedicated placement class rather than in the top-tier placement class. This policy provides a means of reserving certain volumes in a file system's volume set for a preferred class of files, such as database files.

Templates for the four built-in policy types listed in Table 6-1 are part of the Storage Foundation. All four policies relocate files up and down a conceptual hierarchy of between two and five storage tiers (placement classes) based on different relocation qualifiers. The policy types listed in Table 6-1 that use average I/O temperature as a relocation qualifier all use both read and write activity to calculate I/O temperature. All four policies include provision for pinning designated sets of files to either the highest or the lowest placement class in the hierarchy or both.

Launching the built-in policy wizard

Administrators use the VEA graphical management console to launch the VxFS File Placement Policy Wizard to create or modify any of these policies, supplying parameters such as placement class names for the storage tiers and specifications for files to be exempted from policy rules. The wizard generates a placement policy XML document based on the parameters supplied, and when running locally (not under the control of Storage Foundation Management Server), assigns the policy as the file system's active policy. When running under the control of the Management Server, the wizards store the resulting policy in the Management Server database. Figure 6-1 illustrates the use of the VEA console to invoke the file placement policy wizard to create a new file placement policy and assign it as the active placement policy for the selected file system (at mount point `/volset`).

Figure 6-1 Invoking the VEA file placement policy wizard to create a new policy



Name	Size	Free space	Type	Mount p...	Usage
volset	500.000 MB	466.456 MB	vxfs	/volset	1%

The primary pop-up menu in Figure 6-1 lists the actions that can be taken on VxFS file system objects from the VEA console. The second-level menu lists the file placement policy actions, which are described in Table 6-2.

Table 6-2 VEA commands for file placement policies

Command	Description
`Create...`	This command invokes the File Placement Policy Creation Wizard to create one of the four types of built-in policies and assign it as the active policy for the selected file system.
`Modify...`	This command invokes the File Placement Policy Creation Wizard and populates it with the selected file system's current active policy, provided that it is one of the four built-in policy types. (If it is not, the command fails). Using the wizard, an administrator can edit the policy, and assign the edited version to the selected file system as a new active policy.

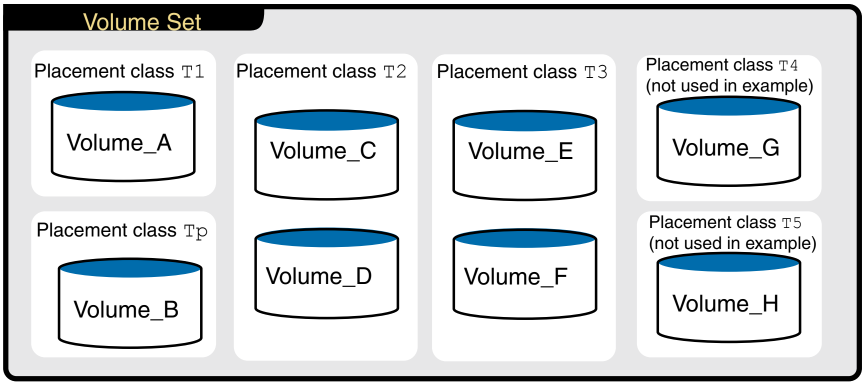
Table 6-2 VEA commands for file placement policies *(continued)*

Command	Description
`Delete...`	This command deletes the selected file system's current active placement policy, whether or not it is one of the four built-in types. This command leaves a file system without a file placement policy, so after it executes, VxFS chooses locations within the volume set for newly created files, and no file relocation occurs.
`Analyze...`	This command is equivalent to the `fsppadm analyze` CLI command. It scans all or part of the selected file system's directory tree against its active placement policy and creates a report describing the actions that would be taken if the `Enforce...` command were invoked. This command does not relocate any files.
`Enforce...`	This command is equivalent to the `fsppadm enforce` console command. It enforces the selected file system's active policy against some or all of its files by scanning the directory tree and relocating or deleting files as indicated by the policy rules.
`Query`	This command produces a report similar to that shown in Dialog 4-8. The report provides file-by-file information on original, current, and policy-directed locations, and the rules under which files will be relocated during the next `Enforce` operation.

An example of built-in policy creation

The remainder of this appendix describes the use of the File Placement Policy Creation Wizard to create an access age-based with preferred files placement policy on a file system with three tiers (placement classes) of volumes for normal files, a fourth tier reserved for preferred files, and two additional tiers that are not used in this example. The file system's volume set and placement class configuration for the example is represented in Figure 6-2.

Figure 6-2 File system for policy creation example



The volume set represented in Figure 6-2 has three tiers of volumes with placement classes `T1`, `T2`, and `T3`. The built-in policy treats these as a three-level hierarchy. Additionally, the policy reserves a fourth placement class called `Tp` as top storage tier for a specified set of preferred files.

The access age-based with preferred files policy allocates new non-preferred files on the `T1` volume if free space permits, on one of the `T2` volumes if not, and on one of the `T3` volumes if neither the `T1` volume nor the `T2` volumes have adequate free space. The policy relocates non-preferred files on the `T1` volume that are inactive for a designated period to one of the `T2` volumes, free space permitting, and to one of the `T3` volumes if the `T2` volumes are filled to capacity. Similarly, the policy relocates files on the `T2` volumes that remain inactive for a (longer) designated period to one of the `T3` volumes, again if free space permits.

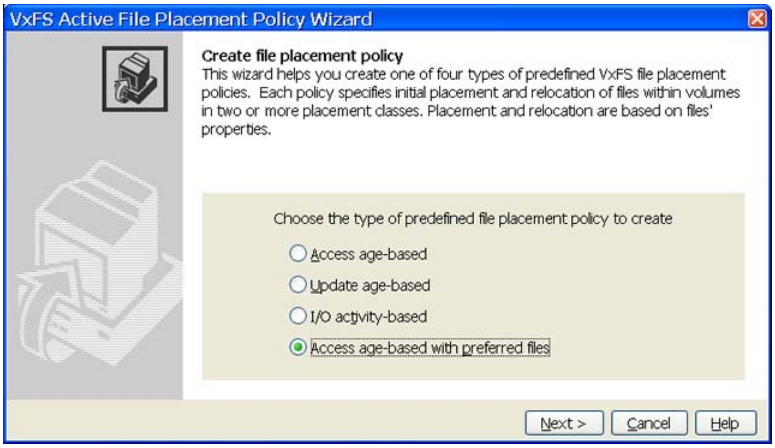
The policy relocates non-preferred files from `T2` and `T3` volumes to the `T1` volume if their average I/O temperatures, calculated using both read and write activity, rise above a specified level. Similarly, it relocates files on `T3` volumes whose temperatures rise to a (lower) designated level to `T2` volumes.

The policy treats preferred files (as designated in a policy rule `SELECT` statement) identically with non-preferred ones, except that initial allocation is on the `Tp` volume rather than the `T1` volume, and I/O temperature-based upward relocation is to the `Tp` volume rather than to the `T1` volume. Under this policy, no non-preferred files are ever allocated on or relocated to the `Tp` volume, and no preferred files are allocated on or relocated to the `T1` volume. Thus, the `Tp` volume is effectively reserved for files specified as preferred.

Choosing the policy type

When the `Create...` command is invoked from the VEA console, VEA displays the introductory wizard page shown in Figure 6-3.

Figure 6-3 Choosing the file placement policy type



Create file placement policy

This wizard helps you create one of four types of predefined VxFS file placement policies. Each policy specifies initial placement and relocation of files within volumes in two or more placement classes. Placement and relocation are based on files' properties.

Choose the type of predefined file placement policy to create

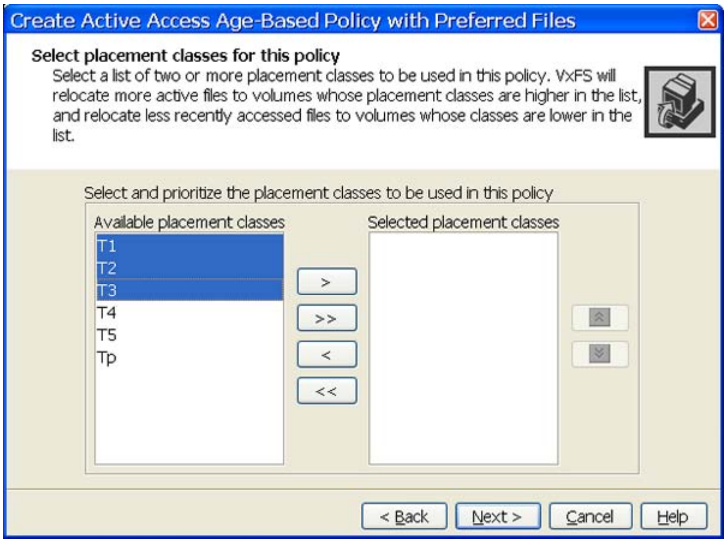
- ○ Access age-based
- ○ Update age-based
- ○ I/O activity-based
- ● Access age-based with preferred files

Administrators use this introductory page to select the type of policy to be created. This example describes the access age-based with preferred files policy because it is the most complex of the four built-in policy types. Wizards for other built-in policy types are similar, with the exception that no provision is made for preferred files.

Choosing applicable storage tiers

When the policy type has been selected and the **Next** button clicked, the placement class selection page shown in Figure 6-4 appears.

Figure 6-4 Choosing storage tiers for a file placement policy


Create Active Access Age-Based Policy with Preferred Files

Select placement classes for this policy

Select a list of two or more placement classes to be used in this policy. VxFS will relocate more active files to volumes whose placement classes are higher in the list, and relocate less recently accessed files to volumes whose classes are lower in the list.

Select and prioritize the placement classes to be used in this policy

Available placement classes: T1, T2, T3, T4, T5, Tp

Selected placement classes

This page is used to select the placement classes to be used by the policy being created. In the case of the access age-based with preferred files policy, this page is used to select the placement classes that are to be used for non-preferred files. Other built-in policies make no distinction for preferred files, so their corresponding pages define the placement classes to be used for all files.

When the placement policy creation wizard is invoked through VEA on the local host, it queries the selected file system's volume set to determine the placement classes it contains. Thus, the list of placement classes that appears in the left panel is the complete list of placement classes in the file system's volume set. Under normal circumstances, all classes except the one to be reserved for preferred files should be selected. (The other three built-in policy types do not have preferred files, so all available placement classes should normally be selected for them.) When policy creation is complete, and the resulting policy is assigned as the file system's active policy, volumes belonging to any placement classes that have not been selected on this page are not used to store files.

When the wizard is invoked through the Storage Foundation Management Server, there is no connection to a particular file system. The list of placement classes displayed in this panel includes all placement classes contained in the Management Server database. Placement classes that are present in the volume sets of file systems to which the policy being created will be applied should be selected in this case.

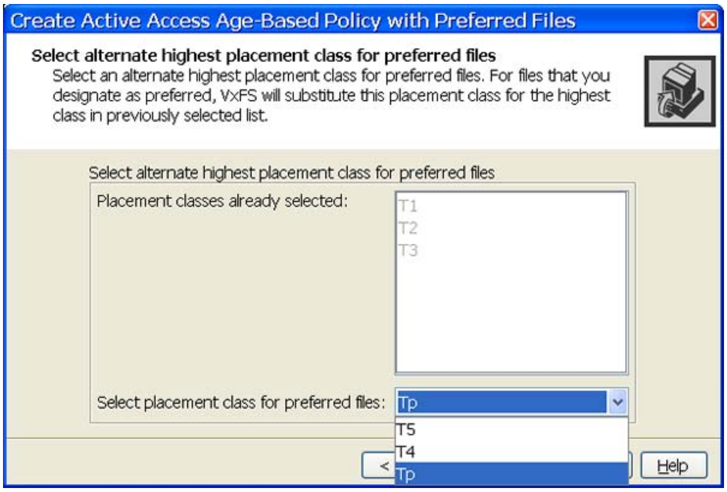
Policies created using the Storage Foundation Management Server

Because file placement policies stored in the Management Server database may be assigned to any file system on any host on which Management Server client software is installed, a file system's active policy may refer to placement classes that are not present in its volume set. When this occurs, the Dynamic Storage Tiering facility treats the missing placement classes as though they were completely occupied. This makes it possible to use Management Server to assign a single policy to multiple file systems, even though not all placement classes named in the policy are present in all file systems' volume sets. For example, if this wizard were run under the control of Management Server, placement classes `T4` and `T5`, although in the database, might not be present on any of the systems to which the policy being created will be applied.

The placement classes selected using the page shown in Figure 6-4 are listed in hierarchical order. The policy created by this wizard treats the placement class listed first as the top (most desirable) storage tier, the second listed class as the second tier, and so forth. The position of a placement class in the selected list may be changed by selecting it and using the two vertical arrow buttons on the right side of the page.

When the placement classes for non-preferred files have been selected and the `Next` button clicked, the page shown in Figure 6-5 appears. This page is used to select the placement class to be reserved as top-tier storage for preferred files.

Figure 6-5 Choosing the placement class for preferred files


Create Active Access Age-Based Policy with Preferred Files

Select alternate highest placement class for preferred files

Select an alternate highest placement class for preferred files. For files that you designate as preferred, VxFS will substitute this placement class for the highest class in previously selected list.

Select alternate highest placement class for preferred files

Placement classes already selected: T1, T2, T3

Select placement class for preferred files: Tp

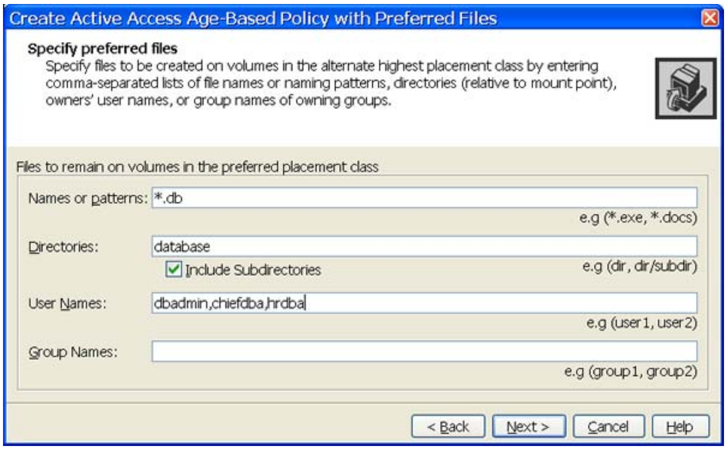
In this example, placement class `Tp` is selected as the top-most storage tier for preferred files, which are designated at a later stage in the policy. As Figure 6-2 shows, the `Tp` placement class represents a single volume in this example. Designating the volume's placement class rather than its name has a powerful advantage, however. Additional volumes can be added to a file system's volume set. If these are tagged as `Tp` volumes, VxFS uses them as top-tier storage for preferred files automatically. No policy change is required.

Volume tags are not unique within a file system's volume set. Any number of volumes can be tagged with the value `Tp`. This makes it possible to use one single policy with a number of similar systems of different capacities. As long as volumes that should be reserved for storing preferred files are tagged `Tp`, the policy produces the desired behavior no matter how many or how few of them there are. This reduces administrative effort and introduces a level of uniformity on data center operations that is likely to improve the quality of service delivered to business applications.

Designating preferred files

The next step in policy creation is to specify which files should be treated as preferred files. Preferred files may be designated by name or pattern, by directory, or by owner name or owner's group name. Figure 6-6 shows the wizard page used to specify the files that the access age-based with preferred files policy treats as preferred.

Figure 6-6 Designating preferred files


Create Active Access Age-Based Policy with Preferred Files

Specify preferred files
Specify files to be created on volumes in the alternate highest placement class by entering comma-separated lists of file names or naming patterns, directories (relative to mount point), owners' user names, or group names of owning groups.

Files to remain on volumes in the preferred placement class

Names or patterns: *.db
e.g (*.exe, *.docs)

Directories: database
✓ Include Subdirectories
e.g (dir, dir/subdir)

User Names: dbadmin,chiefdba,hrdba
e.g (user1, user2)

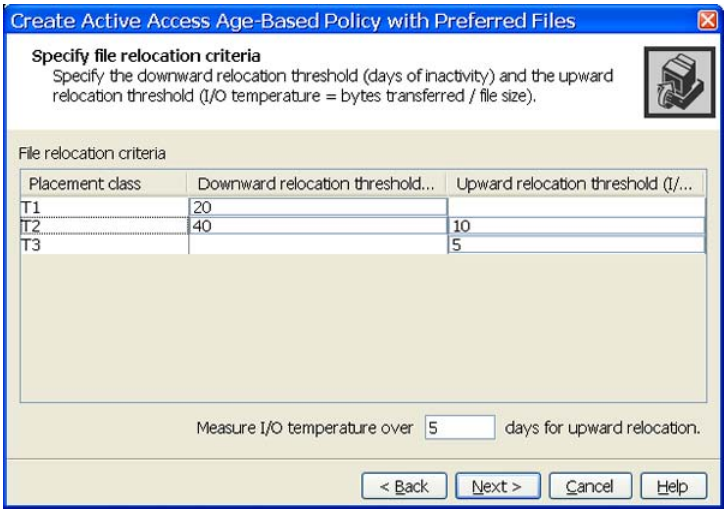
Group Names:
e.g (group1, group2)

Any or all of the four preferred file selection criteria shown in Figure 6-6 may be specified. Multiple values may be specified for any single criterion by separating values with commas. Thus for example in Figure 6-6, files owned by any of users `dbadmin`, `chiefdba`, and `hrdba` may be treated as preferred.

If multiple criteria are specified, a file must satisfy all of them in order to be treated as preferred. In this example, a file must have a name extension of `db`, reside in directory `database` (relative to the file system mount point) or one of its subdirectories, and be owned by one of users `dbadmin`, `chiefdba`, and `hrdba` in order to be eligible for preferred treatment. All files that are not designated as preferred on this page are treated as non-preferred, and are allocated on and relocated to the `T1` volume rather than the `Tp` one.

Defining relocation qualifiers

Figure 6-7 Choosing upward and downward relocation thresholds



Specify file relocation criteria
Specify the downward relocation threshold (days of inactivity) and the upward relocation threshold (I/O temperature = bytes transferred / file size).

File relocation criteria

Placement class	Downward relocation threshold...	Upward relocation threshold (I/...
T1	20	
T2	40	10
T3		5

Measure I/O temperature over 5 days for upward relocation.

The next step in policy definition is to specify the parameters that control relocation of both preferred and non-preferred files using the wizard page shown in Figure 6-7. On this page, the administrator enters the numeric thresholds for downward and upward relocation of files (both non-preferred and preferred) within the previously chosen storage tiers. The middle column contains the number of days of inactivity that trigger downward migration from tier to tier. In Figure 6-7, relocation from the `T1` volume to a `T2` volume after 20 days of inactivity has been specified, as has relocation from `T2` volumes to `T3` volumes after 40 days of inactivity. The wizard requires that the list of downward relocation qualifiers be monotonically increasing. In this example, with three storage tiers, the inactivity level for relocating files from `T2` volumes to `T3` volumes must be greater than that for relocating files from `T1` volumes to `T2` volumes.

The rightmost column of the page is used to specify numeric qualifications for upward relocation. In Figure 6-7, an average I/O temperature of 5 over a period of five days is specified for relocating files on `T3` volumes to `T2` volumes, and an average I/O temperature of 10 over the same period is specified for migrating files on `T2` volumes to the `T1` volume. The wizard requires that the list of numeric qualifications for upward relocation be monotonically decreasing. In this example, the I/O temperature for relocating files from `T3` volumes to `T2` volumes must be less than that for relocating files to `T1` volumes.

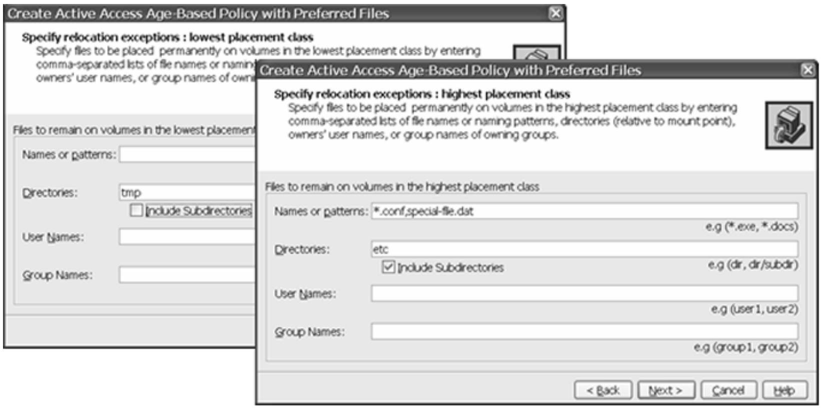
Function vs. simplicity

All four of the VxFS built-in file placement policies restrict functionality in the interest of simplicity. For example, in the access age-based with preferred files policy of this example, both preferred and non-preferred files are relocated on the same schedule—20 and 40 days of inactivity for downward relocation, and average I/O temperatures of 10 and 5 for upward relocation over the same five-day period. The Dynamic Storage Tiering facility policy definition XML grammar supports greater generality, which in the case of built-in policies has been sacrificed in the name of simplicity. The policy definition wizard described in this example produces XML documents which can be edited by a text or XML editor. Once edited, however, the documents can no longer be imported into the wizard for modification by the `Modify...` VEA command.

Pinned files

The final input step in wizard-based built-in policy creation is the optional specification of files to be exempt from relocation. Designated sets of files can be pinned to the top-most and lowest tiers specified on the page illustrated in Figure 6-4, and not subject to relocation under normal circumstances. Figure 6-8 illustrates the wizard pages used to designate files that should remain on the highest and lowest tiers in the hierarchy.

Figure 6-8 Specifying exceptions in the access age-based with preferred files policy



Create Active Access Age-Based Policy with Preferred Files

Specify relocation exceptions : lowest placement class
Specify files to be placed permanently on volumes in the lowest placement class by entering comma-separated lists of file names or namin
owners' user names, or group names of own

Files to remain on volumes in the lowest placemen

Names or patterns:
Directories: tmp
☐ Include Subdirectories
User Names:
Group Names:

Create Active Access Age-Based Policy with Preferred Files

Specify relocation exceptions : highest placement class
Specify files to be placed permanently on volumes in the highest placement class by entering comma-separated lists of file names or naming patterns, directories (relative to mount point), owners' user names, or group names of owning groups.

Files to remain on volumes in the highest placement class

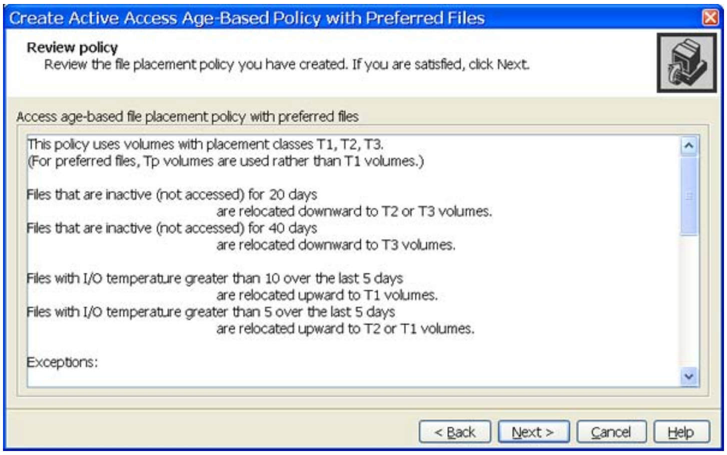
Names or patterns: *.conf,special-file.dat
e.g (*.exe, *.docs)
Directories: etc
☑ Include Subdirectories
e.g (dir, dir/subdir)
User Names:
e.g (user1, user2)
Group Names:
e.g (group1, group2)

The entries in Figure 6-8 specify that files in directory `etc` (and all of its subdirectories, as indicated by the `Include Subdirectories` check box) that are named `special-file.dat` or whose names have the extension `conf` following the period are to be placed on the `T1` volume and not be relocated. Similarly, files in the directory `tmp` (but not its subdirectories) are to be placed on `T3` volumes (the lowest tier specified on the Figure 6-4) and not relocated. The built-in policies produced by these wizards are designed so that if newly-created exempt files must be placed elsewhere than their target placement classes due to lack of free space, they are relocated to their target placement classes when space becomes available on them.

The policy summary

Specification of exempt files concludes the input phase of built-in policy creation. The final step is a review of the policy that the wizard will create based on the input supplied. Figure 6-9 and Figure 6-10 illustrate the policy summary display as presented for review.

Figure 6-9 Summary of built-in policy part 1

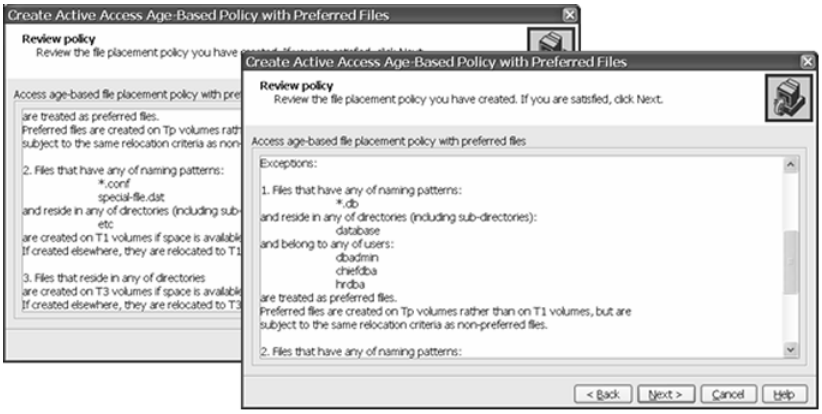

Create Active Access Age-Based Policy with Preferred Files
Review policy
Review the file placement policy you have created. If you are satisfied, click Next.
Access age-based file placement policy with preferred files
< Back
Next >
Cancel
Help

```
This policy uses volumes with placement classes T1, T2, T3.
(For preferred files, Tp volumes are used rather than T1 volumes.)

Files that are inactive (not accessed) for 20 days
                    are relocated downward to T2 or T3 volumes.
Files that are inactive (not accessed) for 40 days
                    are relocated downward to T3 volumes.

Files with I/O temperature greater than 10 over the last 5 days
                    are relocated upward to T1 volumes.
Files with I/O temperature greater than 5 over the last 5 days
                    are relocated upward to T2 or T1 volumes.

Exceptions:
```

Figure 6-9 shows the part of the textual summary that describes the rules for creation and relocation of both non-preferred and preferred files. Each statement in this panel corresponds to a rule in the policy that the wizard creates.

The two views shown in Figure 6-10 illustrate the part of the summary that describes exceptions to the relocation rules in Figure 6-9. These include both preferred files, which are subject to relocation, but which are targeted for the preferred placement class (`Tp` in this example), and files that are pinned to the highest (`T1` in this example) and lowest (`T3` in this example) placement classes respectively. The summary display gives an administrator the opportunity for common-sense validation of the policy that the wizard will save and possibly assign upon completion.

Figure 6-10 Summary of built-in policy part 2


Create Active Access Age-Based Policy with Preferred Files
Review policy
Review the file placement policy you have
Access age-based file placement policy with pre
Create Active Access Age-Based Policy with Preferred Files
Review policy
Review the file placement policy you have created. If you are satisfied, click Next.
Access age-based file placement policy with preferred files
< Back
Next >
Cancel
Help

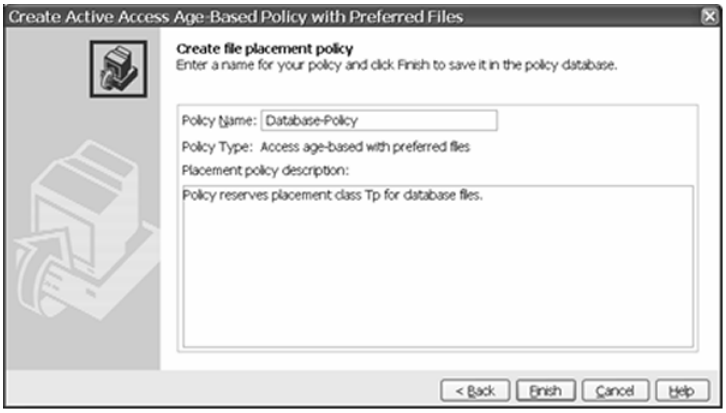
```
are treated as preferred files.
Preferred files are created on Tp volumes rath
subject to the same relocation criteria as non-

2. Files that have any of naming patterns:
                    *.conf
                    special-file.dat
and reside in any of directories (including sub-
                    etc
are created on T1 volumes if space is availabl
If created elsewhere, they are relocated to T1

3. Files that reside in any of directories
are created on T3 volumes if space is availabl
If created elsewhere, they are relocated to T3
```

```
Exceptions:

1. Files that have any of naming patterns:
                    *.db
and reside in any of directories (including sub-directories):
                    database
and belong to any of users:
                    dbadmin
                    chiefdba
                    hrdba
are treated as preferred files.
Preferred files are created on Tp volumes rather than on T1 volumes, but are
subject to the same relocation criteria as non-preferred files.

2. Files that have any of naming patterns:
```

Naming and assignment

The final step in executing the built-in policy wizard, illustrated in Figure 6-11, is to name the policy and assign it to the selected file system (if the wizard is executed through the local VEA interface) or save it in the Storage Foundation Management Server database (if the wizard is invoked through the Management Server framework). The name entered in this step is used to locate the policy for editing when the `Modify...` wizard is invoked. Additionally, a free-form text description of the policy may be supplied. This description is saved with the policy and displayed when the `Query...` command is executed. If the periods specified for average I/O temperature in Figure 6-7 require an increase in the `fcl_keeptime` tunable, checking the box in the lower right corner of the screen permits VxFS to change the value to one large enough to permit `IOTEMP`-based relocations to occur. (The `fcl_keeptime` tunable is discussed in Policies that contain I/O temperature .)

Figure 6-11 Saving and assigning newly-created policy


Create Active Access Age-Based Policy with Preferred Files

Create file placement policy
Enter a name for your policy and click Finish to save it in the policy database.

Policy Name: Database-Policy
Policy Type: Access age-based with preferred files
Placement policy description:
Policy reserves placement class Tp for database files.

Summary: built-in file placement policies

The VxFS built-in file placement policy wizards can construct policies for as few as two or as many as five storage tiers (placement classes). Additionally, the access age with preferred files policy requires a reserved placement class on which preferred files are placed. No matter how many placement classes are specified, any exempt files specified are pinned to the highest and lowest classes (those at the top and bottom of the list specified on the page illustrated in Figure 6-4. Exempt files are optional. If none are specified, all files are subject to relocation according to the numeric qualifications entered on the page shown in Figure 6-7.

The access age-based with preferred files policy illustrated in this example and the access age-based policy both use access age (the interval between a file's POSIX `atime` and the time at which the policy is enforced) as the qualifier for relocating files from higher tiers to lower ones. Both policies use average I/O temperature, calculated using both read and write activity, as their qualifier for upward relocation.

The update age-based built-in policy uses modification age (the interval between a file's POSIX `mtime` and the time at which the policy is enforced) to qualify files for both downward and upward relocation. To eliminate the possibility of endless relocation cycles, the wizard for creating this policy fills in the qualifying modification age values for upward relocation as the administrator supplies values for downward relocation.

The I/O activity-based policy uses average I/O temperature, calculated using both read and write activity, to qualify files for both downward and upward relocation. To eliminate the possibility of endless relocation cycles, the wizard for creating this policy fills in valid qualifying average I/O temperature values for upward relocation as the administrator supplies values for downward relocation.

The four built-in policy types represent a small subset of the full capability of the Dynamic Storage Tiering facility's file placement management capabilities. They do, however, cover a substantial fraction of typical use cases, and can literally be created and assigned to file systems in minutes. The policy documents they produce can be edited to add capabilities not provided by the built-in policies themselves, but once edited by a text editor, the documents cannot be edited using the `Modify...` menu command from the VEA console.

Part II

The applications: using the Dynamic Storage Tiering facility

Using Dynamic Storage Tiering: file relocation based on I/O activity

This chapter includes the following topics:

The value proposition revisited

Perhaps the most broadly applicable benefit of the VxFS Dynamic Storage Tiering facility is reduction of average online storage cost by relocating infrequently accessed files to the less-expensive tiers of storage in a multi-tier hierarchy. Inactivity-based file relocation is predicated on two assumptions about storage cost and data value that are listed in Table 7-1.

Table 7-1 Storage and data assumptions for inactivity-based file relocation

Assumption	Description
Unit cost of storage	Storage in a lower tier in the hierarchy has significantly (50% or more) lower per-byte cost than storage in the next higher tier.

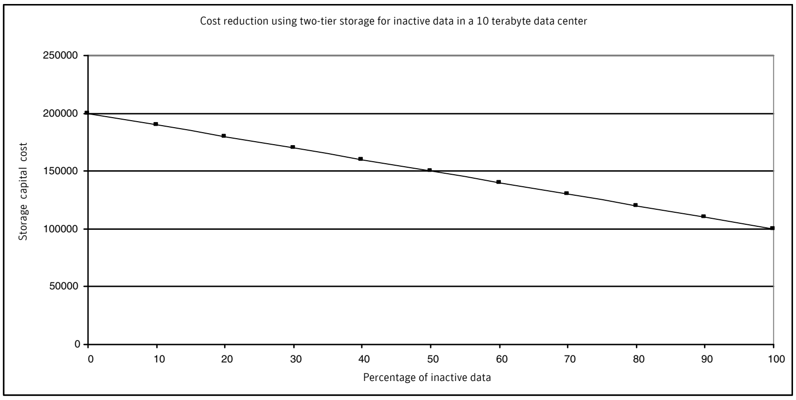
Table 7-1 Storage and data assumptions for inactivity-based file relocation *(continued)*

Assumption	Description
Consequences of unavailable data	The potential consequences of storing infrequently accessed data on lower-performance, less fault-tolerant devices are longer recovery times and older recovery points. These consequences are outweighed by the cost savings from deploying less expensive storage hardware.

The unit cost of storage

The cost differential between different types of online storage creates the economic justification for multi-tier storage strategies. For example, if top-quality enterprise storage hardware costs $20 per gigabyte and mid-range storage costs $10 per gigabyte, an enterprise whose online data is 50% inactive could save 25% of its storage acquisition cost by keeping its inactive data on mid-range storage. Larger percentages of inactive data result in greater savings, as Figure 7-1 illustrates.

Figure 7-1 Potential capital cost reduction from using two-tier storage

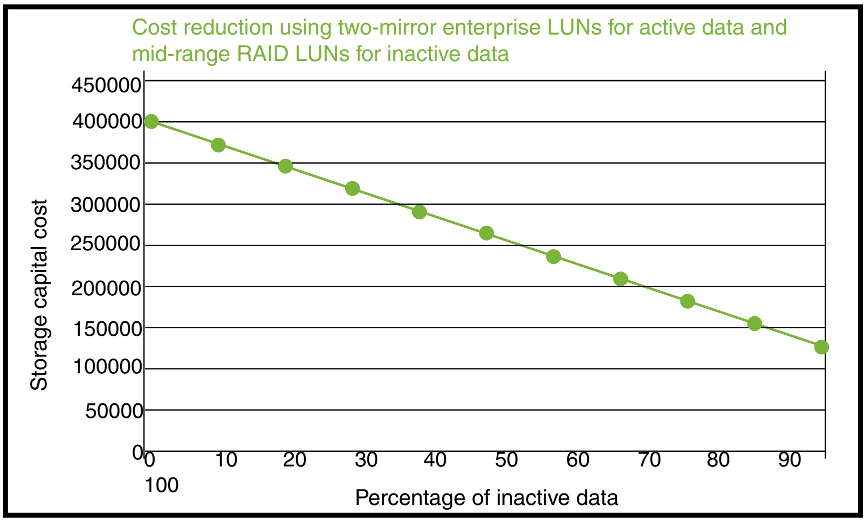


Of course no enterprise actually purchases its entire complement of storage hardware capacity at one time. The graph in Figure 7-1 might more properly be thought of as a continuum of possible end states resulting from a process of gradually adding second-tier storage hardware and configurations and reclaiming top-tier storage by relocating inactive data to second tier volumes. Second-tier storage can be added by purchasing mid-range disk arrays, or by reconfiguring existing unused capacity, for example by replacing mirrored LUNs with RAID-5 ones.

Figure 7-1 represents a pure storage hardware-based cost comparison for a data center with 10 terabytes of online storage. Volume (LUN) configurations are assumed to be the same for both first and second tier hardware. In reality, the typical cost differential is even greater due to configuration differences. The mirrored LUNs typically configured on enterprise-class arrays require more storage hardware per usable byte, but protect against more failure modes and provide faster recovery than the RAID-5 or simple LUN configurations that tend to be found on mid-range arrays. Mirrored LUNs are commonly configured for an enterprise's most critical data, while RAID-5 or non-redundant LUNs serve less-critical applications. Differences like these in configuration accentuate the potential savings from a multi-tier storage strategy.

For example, using the same basic parameters as Figure 7-1 (10 terabytes of usable data, $20 per gigabyte for enterprise storage hardware, $10 per gigabyte for mid-range storage hardware), configuring enterprise storage as two-mirror LUNs (100% overhead) and mid-range storage as five-disk RAID-5 LUNs (25% overhead) would result in the cost differential curve shown in Figure 7-2.

Figure 7-2 Potential capital cost reduction using two-tier storage in mirrored and RAID-5 configurations



As Figure 7-2 demonstrates, a not uncommon 75% of infrequently accessed data results in storage hardware cost savings of 50% under these configuration assumptions. Taking into account typical enterprise practices of making multiple copies of critical data, the storage hardware savings from placing less critical data on volumes that cost less, both because of the hardware that implements them and the way they are configured, can be even more dramatic.

The value of available data

The second value premise of multi-tier storage is that storing inactive data on storage devices of lesser quality does not affect IT operations adversely. From an I/O performance standpoint, if data is inactive or seldom-accessed, the performance of the storage devices that hold it is irrelevant. From an availability standpoint, it is practical to recover inactive data from backup copies, because time to restore from an offline backup does not delay operations if the files being restored are largely inactive.

In summary, for enterprises that must keep a significant amount of non-critical data online, a multi-tier storage strategy can offer substantial cost savings without adverse effects on business operations.

The multi-tier storage challenge

The challenge in attaining the benefits of multi-tier storage is to get the right files on the right storage tier at the right times. Capital savings can easily be overwhelmed by the administrative cost and complexity of relocating application data and the increased chances of operational error that accompany complexity. The purpose of the Dynamic Storage Tiering facility is to eliminate any administrative cost and complexity by automating the relocation of files as levels of I/O activity against them rise and fall, as well as when their sizes, owners, or logical positions in the file system name space change.

Limitations of conventional techniques

The two conventional techniques for utilizing multi-tier storage are ad hoc data relocation using administrator-defined and maintained procedures, and pre-packaged hierarchical storage management. Both of these have limitations, as Part I (Conventional multi-tier storage techniques) explains. Ad hoc relocation is labor-intensive and susceptible to changes in the IT environment. Hierarchical storage management policies are inflexible, and in addition, there is an inherent time-to-first-byte latency that makes HSM unsuitable for use with many applications. Moreover, when HSM is in use, administrators must ensure that file systems have adequate free space for restoration of migrated files when applications access them. The limitations of both techniques stem from the fact that they relocate files logically as well as physically—file data is moved outside the storage space controlled by the file system. With ad hoc data relocation, applications and operating procedures must be set up to access files in alternate file systems, and the setup must be changed each time files' logical locations change. HSM leaves "stubs" of relocated files in the file system name space, but must retrieve file data from backing store when applications access files.

The VxFS Dynamic Storage Tiering facility has neither of these limitations. Files relocated by DST remain on volumes controlled directly by the file system. When accessed by applications, they are instantly available with no time-to-first-byte latency. Except for any noticeable I/O performance differences between storage tiers, DST file relocation is transparent to applications.

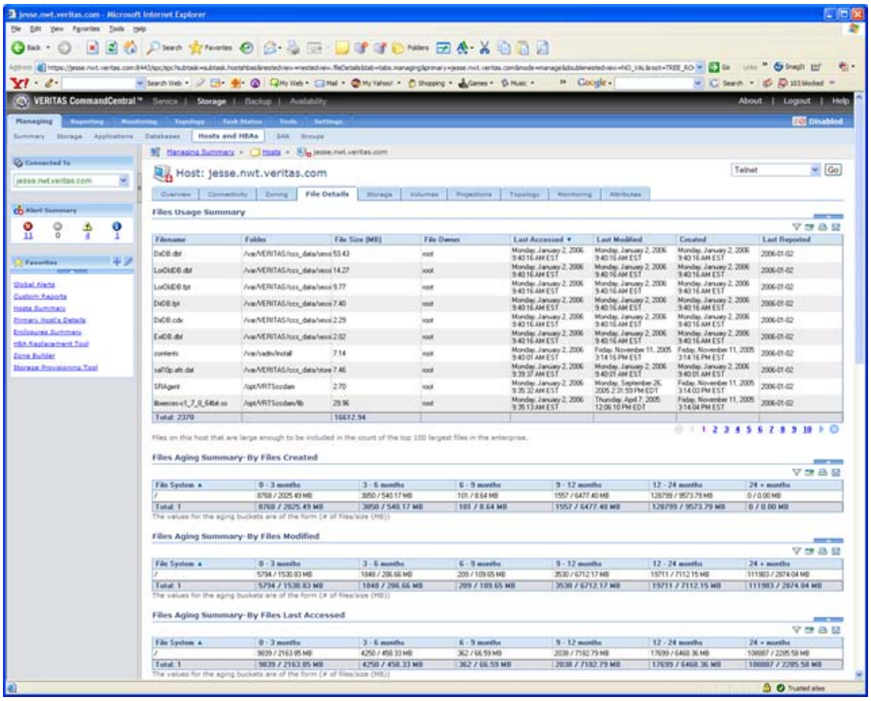
Matching policies to usage

To derive maximum benefit from the Dynamic Storage Tiering facility, a file system's placement policy should match both application usage of files and available storage resources. For example, if 25% of the storage managed by a file system is occupied by files deemed active by some measure, and the rest by inactive files, then 25% of the file system's storage resources should be purchased and configured to meet the I/O performance and data availability needs of the active data. The remaining 75% can be purchased and configured to minimize cost.

Discovering file characteristics

The challenge lies in determining what constitutes an inactive file for purposes of relocation by the DST facility. For some applications, this may be obvious—last month's transaction files are relocated at the end of the month, for example. But in most file systems, setting a file placement policy should be preceded by an analysis of actual file system activity. The Command Central Storage component of the Storage Foundation can generate file utilization reports that provide some guidance on configuring storage to match utilization patterns, as Figure 7-3 illustrates.

Figure 7-3 Command Central Storage file activity summary



Host: jesse.nwt.veritas.com

Files Usage Summary

Filename	Folder	File Size (MB)	File Owner	Last Accessed ▼	Last Modified	Created	Last Reported
DiDB.dbf	/var/VERITAS/ccs_data/sessi	53.43	root	Monday, January 2, 2006 9:40:16 AM EST	Monday, January 2, 2006 9:40:16 AM EST	Monday, January 2, 2006 9:40:16 AM EST	2006-01-02
LoOldDB.dbf	/var/VERITAS/ccs_data/sessi	14.27	root	Monday, January 2, 2006 9:40:16 AM EST	Monday, January 2, 2006 9:40:16 AM EST	Monday, January 2, 2006 9:40:16 AM EST	2006-01-02
LoOldDB.tpr	/var/VERITAS/ccs_data/sessi	9.77	root	Monday, January 2, 2006 9:40:16 AM EST	Monday, January 2, 2006 9:40:16 AM EST	Monday, January 2, 2006 9:40:16 AM EST	2006-01-02
DiDB.tpr	/var/VERITAS/ccs_data/sessi	7.40	root	Monday, January 2, 2006 9:40:16 AM EST	Monday, January 2, 2006 9:40:16 AM EST	Monday, January 2, 2006 9:40:16 AM EST	2006-01-02
DiDB.cdx	/var/VERITAS/ccs_data/sessi	2.29	root	Monday, January 2, 2006 9:40:16 AM EST	Monday, January 2, 2006 9:40:16 AM EST	Monday, January 2, 2006 9:40:16 AM EST	2006-01-02
ExDB.dbf	/var/VERITAS/ccs_data/sessi	2.02	root	Monday, January 2, 2006 9:40:16 AM EST	Monday, January 2, 2006 9:40:16 AM EST	Monday, January 2, 2006 9:40:16 AM EST	2006-01-02
contents	/var/sadm/install	7.14	root	Monday, January 2, 2006 9:40:01 AM EST	Friday, November 11, 2005 3:14:16 PM EST	Friday, November 11, 2005 3:14:16 PM EST	2006-01-02
saf0p.aft.dat	/var/VERITAS/ccs_data/store	7.46	root	Monday, January 2, 2006 9:39:37 AM EST	Monday, January 2, 2006 9:40:01 AM EST	Monday, January 2, 2006 9:40:01 AM EST	2006-01-02
SRAgent	/opt/VRTSccsdam	2.70	root	Monday, January 2, 2006 9:35:32 AM EST	Monday, September 26, 2005 2:31:59 PM EDT	Friday, November 11, 2005 3:14:03 PM EST	2006-01-02
libxerces-c1_7_0_64bit.so	/opt/VRTSccsdam/lib	29.96	root	Monday, January 2, 2006 9:35:13 AM EST	Thursday, April 7, 2005 12:06:10 PM EDT	Friday, November 11, 2005 3:14:04 PM EST	2006-01-02
Total: 2370		16612.94					

1 2 3 4 5 6 7 8 9 10

Files on this host that are large enough to be included in the count of the top 100 largest files in the enterprise.

Files Aging Summary-By Files Created

File System ▲	0 - 3 months	3 - 6 months	6 - 9 months	9 - 12 months	12 - 24 months	24 + months
/	8768 / 2025.49 MB	3850 / 540.17 MB	101 / 8.64 MB	1557 / 6477.40 MB	128799 / 9573.79 MB	0 / 0.00 MB
Total: 1	8768 / 2025.49 MB	3850 / 540.17 MB	101 / 8.64 MB	1557 / 6477.40 MB	128799 / 9573.79 MB	0 / 0.00 MB

The values for the aging buckets are of the form (# of files/size (MB))

Files Aging Summary-By Files Modified

File System ▲	0 - 3 months	3 - 6 months	6 - 9 months	9 - 12 months	12 - 24 months	24 + months
/	5794 / 1530.83 MB	1848 / 206.66 MB	209 / 109.65 MB	3530 / 6712.17 MB	19711 / 7112.15 MB	111983 / 2874.04 MB
Total: 1	5794 / 1530.83 MB	1848 / 206.66 MB	209 / 109.65 MB	3530 / 6712.17 MB	19711 / 7112.15 MB	111983 / 2874.04 MB

The values for the aging buckets are of the form (# of files/size (MB))

Files Aging Summary-By Files Last Accessed

File System ▲	0 - 3 months	3 - 6 months	6 - 9 months	9 - 12 months	12 - 24 months	24 + months
/	9839 / 2163.85 MB	4250 / 458.33 MB	362 / 66.59 MB	2038 / 7182.79 MB	17699 / 6468.36 MB	108887 / 2285.58 MB
Total: 1	9839 / 2163.85 MB	4250 / 458.33 MB	362 / 66.59 MB	2038 / 7182.79 MB	17699 / 6468.36 MB	108887 / 2285.58 MB

The values for the aging buckets are of the form (# of files/size (MB))

Figure 7-3 illustrates the first few summary lines of the Command Central Storage `File Details` tab for a single server. This page summarizes file creation, access, and modification activity, file sizes, and file types for a file system. The report contains both the number of files that fit the various categories and the amount of storage capacity they occupy. Almost at a glance, this report can help determine appropriate file selection criteria, as well as relocation qualifiers, that match both actual file utilization patterns and the storage complement of a multi-volume file system.

If Command Central Storage is not available, one analysis technique is to take periodic listings of all files in a file system, sort them by POSIX `atime` or `mtime`, and from the distribution of results, determine the appropriate access age to be part of a downward relocation policy rule. For example, if analysis shows that 75% of files have not been accessed for 30 days or more, 30 days may be a suitable downward relocation threshold.

This analysis can also be used to assess the adequacy of the storage devoted to a file system. For example, if 25% of files are judged to be active (for example, have been accessed within the last 30 days), but those 25% comprise only 10% of the storage capacity occupied by the file system, a redistribution of storage resources in the data center may be appropriate. Conversely, if the active files comprise a greater percentage of total file system storage capacity than is available on top-tier storage, then some active files necessarily occupy second-tier storage, and an increase in the amount of top-tier storage is probably warranted.

Using file activity analysis to define placement policies

If analysis shows a single sharp delineation point between inactive and active files, the same delineation point may also be used in a policy rule for upward relocation. Fragment 7-1 illustrates upward and downward file relocation statements that might have been based on such an analysis of file activity.

Fragment 7-1 Activity-based policy rules

```
<RELOCATE>
  <FROM><SOURCE><CLASS>top_tier</CLASS></SOURCE></FROM>
  <TO><DESTINATION><CLASS>second_tier</CLASS></DESTINATION></TO>
  <WHEN>
    <ACCAGE Units="days"><MIN Flags="gt">30</MIN></ACCAGE>
  </WHEN>
</RELOCATE>

<RELOCATE>
  <FROM><SOURCE><CLASS>second_tier</CLASS></SOURCE></FROM>
  <TO><DESTINATION><CLASS>top_tier</CLASS></DESTINATION></TO>
  <WHEN>
    <ACCAGE Units="days"><MAX Flags="lteq">30</MAX></ACCAGE>
  </WHEN>
</RELOCATE>
```

Time since last access (`<ACCAGE>`) may not always be a suitable qualifier for file relocation. For example, an application may open a file and read a few bytes, determine that the file is not of interest, and close it again. If the application runs daily, the file's access age is always less than a day, so it would never be relocated if the applicable policy rule were similar to that shown in Fragment 7-1, even though there is little reason for it to remain on top tier storage.

Conversely, a file that has been relocated to second-tier storage may be accessed once on the day before a policy enforcement, with no intention of accessing it again for months. Again, such a file's access age is small, so it is likely to be relocated upward if the rule for upward relocation is `<ACCAGE>`-based.

Neither of these results is desirable. The problem with them stems from the fact that access age is sometimes inadequate as an indicator of file activity. Any access to a file effectively zeros its access age. The DST average I/O temperature relocation qualifier is often a better indicator of consistent I/O activity against a file over a period of time.

Policies that contain I/O temperature

Using the `<IOTEMP>` or `<ACCESSTEMP>` relocation qualifiers can reduce or eliminate the undesirable effects of `<ACCAGE>`-based relocation in file systems that are susceptible to them. VxFS computes a file's average I/O temperature or access temperature based on I/O statistics records that are periodically written to its file system's File Change Log (FCL). If a file system's active placement policy includes `RELOCATE` statements that use `<IOTEMP>` as a relocation qualifier, the FCL and I/O statistics collection must both be activated, and moreover, the `fcl_keeptime` must be larger than the largest `<PERIOD>` specified in any policy rule. If a file system's placement policy contains one or more `<IOTEMP>` or `<ACCESSTEMP>` relocation qualifiers, and the FCL is not active when the policy is enforced, no I/O temperature-based file relocations are performed.

When a file placement policy is assigned to a file system, VxFS automatically activates both the FCL and the collection of I/O statistics. If the FCL is deactivated for any reason, it must be reactivated again in order for policy enforcement to perform temperature-based relocations. Dialog 7-1 illustrates a command sequence that can be issued by an authorized administrator to reactivate the FCL, enabling the I/O statistics collection required to enforce I/O temperature based relocation rules, and increasing the maximum time for which FCL records are retained.

Dialog 7-1 Enabling File Change Log statistics collection

```
[1]      # fcladm on /vsa
[2]      # fcladm set filestats /vsa
[3]      # vxtunefs -o fcl_keeptime=432000 /vsa
[4]      # vxtunefs -o fcl_winterval=12h /vsa
```

The `fcladm on` command in Dialog 7-1 starts File Change Log data collection on the file system mounted at `/vsa`. VxFS does not store I/O activity statistics persistently in the FCL by default, so the `fcladm set filestats` command in line `[2]` is required to start collection. If required, the `vxtunefs` command in line `[3]` extends the time for which FCL records are retained so that `<IOTEMP>` and `<ACCESSTEMP>` relocation and deletion qualifiers can be calculated.

When `filestats` storage is enabled, records written to the FCL include information about the number of bytes read from and written to files between the times that their inodes enter and leave the file system cache. VxFS also writes periodic statistics-only records for files that remain open for long periods during which no other FCL records are written on their behalf. The period is adjustable by adjusting the `fcl_winterval` file system tunable as in line `[4]` of Dialog 7-1. Because the DST average I/O temperature computation requires only the oldest and newest I/O statistics records in the FCL, a `fcl_winterval` that is about 25% of the interval between regularly scheduled policy enforcements is usually adequate. For example, if policy enforcement is scheduled daily, a`fcl_winterval` value of 6 hours is nornmally adequate.

Average I/O temperature and access temperature are application-dependent file relocation qualifiers. Application knowledge is required to determine, for example, whether bytes read, bytes written, or both are most appropriate for temperature calculations, what the optimal `<PERIOD>` for measuring temperature is, and what constitutes an appropriate temperature to justify the I/O load of relocating large files. For example, files containing event history for a customer relationship management system might be appropriately located on top-tier storage if they are read or written five times or more per day (an average I/O temperature of 5), and more appropriately moved to second-tier storage if accessed less frequently. In other instances, files might only merit relocation to top-tier storage if they are being modified frequently, so only write statistics would figure in temperature calculation.

Unlike access and modification age, which use only file system metadata and require less application-specific knowledge, temperature-based relocation may require awareness of how users and applications access files, and even some experimentation to determine the most appropriate values for `<PERIOD>`, `<MIN>`, and `<MAX>`.

Modification age-based relocation

The third activity-related file relocation qualifier is modification age (`<MODAGE>`), computed as the difference between a file's POSIX `mtime` and the time at which the active policy is enforced. Modification age is an appropriate relocation qualifier when updates to a file rather than read access make it appropriate to keep it on a top storage tier. This might be the case, for example, if protecting recently modified files from loss due to storage device failure is critical, but read performance of unmodified files is not. As another example, if modification of a previously inactive file is a signal of upcoming regular accesses, a small modification age might qualify a file for relocation from a lower storage tier to a higher one.

A policy may use different qualifiers for relocation from lower storage tiers to higher ones than are used to relocate files from higher tiers to lower. Two of the four file placement policy types built into the Storage Foundation do this, using minimum access age to qualify files for relocation from higher storage tiers to lower ones, and average I/O temperature as the qualifier for relocation from lower tiers to higher. Another combination of qualifiers that may be appropriate for some applications is access age as a downward relocation qualifier and modification age for upward relocation.

IOTEMP and ACCESSTEMP vs. ACCAGE and MODAGE

Average I/O temperature and access temperature are "softer" qualifiers for activity-based relocation than access or modification age. If minimum access or modification age is used to qualify files for downward relocation, a single access to a file between successive policy enforcements prevents relocation. Conversely, using maximum access age as an upward relocation qualifier means that a single access to an inactive file qualifies it for relocation, even if it is never accessed again. For a file to be relocated based on its average I/O temperature, it must experience more than a minimum (for upward relocation) or less than a maximum (for downward relocation) level of I/O activity over a period of time. Both minimum and maximum thresholds, as well as the period over which they are observed can be set to meet application and operational requirements.

Using Dynamic Storage Tiering: efficient backup of large numbers of inactive files

This chapter includes the following topics:

File systems with millions of files

Some file systems are characterized by very large numbers (millions) of small files, most of which eventually become inactive. In many cases, inactive files can only be identified by examining their POSIX `atime`s, and not by other properties, such as the directories in which they reside, or their owners. The "working set" of such file systems is relatively small—10-20% of the total number of files. A mail server that stores each message as a file, a document management system, or any database application using opaque data objects stored as separate files might all have this characteristic.

Challenges in backing up large numbers of inactive files

Such file systems can typically benefit greatly from two-tier storage, with files being relocated to the lower of the two tiers based on one of the four inactivity qualifiers (access age, modification age, average I/O temperature, and average access temperature). If 80-90% of the files are inactive, then 80-90% of the file system's storage could consist of low-cost devices. But there is a further important benefit as well. Backing up a file system with many files (millions or tens of millions) is time-consuming, with much of the time consumed in the processing of inactive file metadata. If the majority of files are inactive, not only is processing time wasted during backup, but I/O time and media are consumed needlessly as well, making copies of inactive files that have not changed since the previous backup cycle.

Advantages of volume-level backup

Properly configured, the Dynamic Storage Tiering facility can facilitate a significant reduction in the time and resources required to back up file systems of this type for disaster recovery. The reduction results from a scheme predicated upon the practicality of making volume-level backups of individual volumes in the file system's volume set (hence the limit of its applicability to disaster recovery: volume-level backups are generally unsuitable for individual file recovery).

The essence of the scheme is to make less frequent image backups of volumes that contain inactive data, and more frequent image backups of volumes that contain active data. This saves time and resources in two ways. First, inactive data need not be backed up as frequently as active data. For example, a schedule of daily backups of active data and weekly backups of inactive data can save 85% of the time and media consumed in backing up the inactive portion of a file system's data. Second, volume-level backup with efficient large I/O requests and zero overhead for metadata processing can be significantly faster than file-level backup for very large numbers of files (assuming that a large percentage of volume capacity is occupied by data so that more data than unoccupied space is backed up).

To use this scheme, second-tier volumes on which inactive data is to be stored must be designated as `dataonly` volumes so that no VxFS metadata is placed on them. All file system metadata will reside on top-tier volumes, along with active data. During normal operation, second-tier volumes are made read-only, except during file placement policy enforcement, when they are made read-write so that inactive files can be relocated to them.

At any time while second-tier volumes are read-only, image backups of them can be made, for example using the UNIX `dd` utility program. Applications can read data on second-tier volumes during backup, but because the volumes are read-only, data on them cannot change. Because second-tier volumes are `dataonly`, all metadata that describes the files stored on them remains online along with the file system's active data on top-tier volumes.

Restoring data from a volume-level backup

If a second-tier volume fails, a replacement volume of the same capacity can be added to the file system's volume set at the same relative position, and the backup image restored onto it. As long as the failed volume remained write-locked from the time of the backup until the time of the restore, data blocks on the restored image will be properly mapped by the metadata on the file system's top-tier volumes.

This scheme requires that second-tier volumes be made writable periodically so that files on top-tier volumes that have become inactive can be relocated to them. During policy enforcement, no valid backup exists, because the data on second-tier volumes changes as newly inactive files are relocated to them. It is important, therefore, that the window during which inactive files are writable be as short as possible. Ideally, second-tier volumes should be made writable immediately before the policy is enforced, and made read-only again as soon as enforcement completes. To provide a measure of protection against data loss during this window, the volume level backups should be augmented by periodic file level backups.

A variation of this scheme incorporating multiple second-tier volumes can shorten backup time for inactive data by making it possible to make multiple image backups concurrently. With multiple second-tier volumes, several `dd`-type copy operations can be run concurrently, shortening elapsed time for backup of second-tier data.

Characteristics of the two-tier volume level backup technique

The volume-level backup scheme has the greatest benefit for file systems that are well-populated (so that volume-level backups back up a preponderance of user data rather than a preponderance of unoccupied space), and file systems in which a large percentage of the files are inactive and have been relocated to second-tier storage before backup runs.

The key benefits of volume-level backup of multi-volume file systems containing large numbers of inactive files are summarized in Table 8-1.

Table 8-1 Benefits of volume-level backup of large numbers of inactive files

Benefit	Description
Significantly reduced backup time and resources compared to full file system backups	Backup time and resource savings result from the higher speed of volume-level (compared to file-level) backup as well as from the elimination of redundant backups of unchanging inactive data. Additionally, backup media and library slot costs are reduced because inactive data can be backed up less frequently. The larger the percentage of inactive files in a file system, the more effective this volume-level backup scheme will be in saving backup time, bandwidth, backup media and online storage cost.
Reduced online storage cost without sacrificing data integrity	Identifying inactive data makes it possible to use lower-cost storage resources for the majority of a file system's storage. The application consequences of waiting for a failed volume containing only inactive data to be restored are generally minimal.
Universal applicability	As long as administrators can manage individual volume backups, the volume-level backup scheme does not require explicit support from backup managers.

Quantitative benefits

Table 8-2 presents a simple example of the volume-level backup scheme that quantifies the cost savings for a one terabyte file system formatted on ten 100 gigabyte volumes under a range of inactive data assumptions.

Table 8-2 Simple benefit analysis for differential volume backup schedules

Number of active data volumes	Number of `dataonly` volumes	Online storage cost	Backup traffic (TB/week)	Elapsed backup time (hr/week)	Media consumed (tapes/week)
10	0	$50,000	7	70	70
9	1	$46,800	6.4	64	64
8	2	$43,600	5.8	58	58
7	3	$40,400	5.2	52	52
6	4	$37,200	4.6	46	46
5	5	$34,000	4	40	40
4	6	$30,800	3.4	34	34

Table 8-2 Simple benefit analysis for differential volume backup schedules *(continued)*

Number of active data volumes	Number of `dataonly` volumes	Online storage cost	Backup traffic (TB/week)	Elapsed backup time (hr/week)	Media consumed (tapes/week)
3	7	$27,600	2.8	28	28
2	8	$24,400	2.2	22	22
1	9	$21,200	1.6	16	16

The example in Table 8-2 is based on the assumptions listed in Table 8-3

Table 8-3 Assumptions for per-volume backup example

Assumption	Description
File system volume configuration	The file system occupies ten 100-gigabyte volumes. The volumes are either based on enterprise-class mirrored LUNs with a raw hardware cost of $25 per gigabyte, or mid-range five-disk RAID-5 LUNs with a raw hardware cost of $15 per gigabyte.
Volume usage	Active files are stored on top-tier mirrored enterprise volumes; less active files are stored on second-tier mid-range RAID-5 volumes.
Backup frequency and timing	Active data volumes are backed up daily; inactive data volumes are backed up weekly. Each backup of a volume takes an hour.

Table 8-2 demonstrates that the savings in raw storage hardware, I/O bandwidth consumed by backup, time during which application I/O performance is degraded (because backup is running), and tape media consumption are all proportional to how much of the file system's data is inactive and storage on low-cost volumes. Storage hardware cost varies by more than a factor of 2, and other metrics vary by more than a factor of 4.

Using Dynamic Storage Tiering: placing data for business reasons

This chapter includes the following topics:

Mapping business requirements to file characteristics

While relocating files among storage tiers in a cost, availability, and performance hierarchy is the most obvious application for the Dynamic Storage Tiering facility, the generality of DST architecture makes other uses possible as well. Table 9-1 lists three additional ways in which the flexibility of DST architecture makes precise file placement based on a variety of business purposes possible.

Table 9-1 Dimensions of flexibility in DST file placement

Dimension of flexibility	Description
Placement classes	VxFS imputes no intrinsic meaning to placement classes. A placement class name is an arbitrary character string attached to volumes in a file system's volume set, and used to identify new file allocation sites as well as the source and destination volumes for file relocation.
File selection criteria	The four file selection criteria (`<PATTERN>`, `<DIRECTORY>`, `<OWNER>`, and `<GROUP>`) provide a variety of ways of organizing files so that appropriate rules can be applied to them. Three of these criteria, `<OWNER>`, `<GROUP>`, and `<DIRECTORY>`, are also commonly used to organize files by application or user, making it natural to dedicate volumes to data used by particular applications or owned by particular users. The fourth criterion (`<PATTERN>`) is commonly used by applications to identify files, making it possible to discriminate in favor of or against files of certain types.
File relocation qualifiers	Files can be relocated or deleted based on activity level, but also on the basis of size, making it possible to relegate very large files to high-capacity storage devices and segregating them from other files in the file system.

The sections that follow illustrate how these properties make it possible to locate files on volumes to satisfy a range of business purposes.

File placement by application

Many enterprises use database management systems to manage their most critical digital data. A reasonable policy for an enterprise's most critical data is to give it preferential placement compared to other files in a file system. An enterprise may have many databases processed by different servers, but there is a strong likelihood of all database-related files being named according to common patterns (usually file name extensions) that are dictated or at least strongly influenced by the database management system. These patterns can be used to control the placement and relocation of database files by specifying them in the `SELECT` statement of a file placement policy rule. The `Database_Rule` in lines `[1]-[8]` of Fragment 9-1 illustrates the selection of database-related files for preferential treatment based on their naming patterns. Preferential treatment might include allocation on top-tier storage, specification of the `any` flag in the `<ON>` clause of a `CREATE` statement, and unconditional upward relocation (so that if a preferred file is created on other than top-tier storage due to lack of available top-tier space, it is relocated to the top tier when space becomes available). Because the `Database_Rule` occurs first in the policy, database files are placed according to it rather than rules that appear later, even if they satisfy the selection criteria in those rules.

Fragment 9-1 Using file naming patterns to control placement

```
[1]      <RULE Name="Database_Rule">
[2]        <SELECT>
[3]          <PATTERN>*.db</PATTERN>
[4]          <PATTERN>*.ora</PATTERN>
[5]          <PATTERN>*.oralog</PATTERN>
[6]        </SELECT>
[7]          CREATE, RELOCATE, and DELETE statements for databases
[8]      </RULE>

[9]      <RULE Name="Undesired_Files_Rule">
[10]       <SELECT>
[11]         <PATTERN>*.jpg</PATTERN>
[12]         <PATTERN>*.mp3</PATTERN>
[13]       </SELECT>
[14]         CREATE, RELOCATE, and DELETE statements for undesired files
[15]     </RULE>

[16]     <RULE Name="Ordinary_Files_Rule">
[17]       <SELECT><PATTERN>*</PATTERN></SELECT>
[18]        CREATE, RELOCATE, and DELETE statements for ordinary files
[19]     </RULE>
```

Name patterns that identify the applications that process a file can also be used to discriminate against files, as the `Undesired_Files_Rule` in lines `[9]-[15]` of Fragment 9-1 illustrates. This rule might relegate selected files to a low tier in the storage hierarchy, and might also constrain the space that could be devoted to such files by dedicating a placement class of limited capacity to them and not specifying the `any` flag in the `<ON>` clause of its `CREATE` statement.

The rule in lines `[16]-[19]` of Fragment 9-1 is the “catchall” rule that specifies placement for all files not selected by rules occurring earlier in the policy. The four policy types built into VxFS follow this sieve structure—the preferred files rule appears first (in the access age with preferred files policy), rules governing files exempted from relocation appear next, and the rule governing all other files appears last in the policy.

File placement by user and department

For file systems that are organized along user lines rather than application ones, the `<OWNER>` and `<GROUP>` selection criteria make it possible to segregate files along departmental or individual user lines. A large file server, for example, might be shared by engineering, sales, marketing, service, and finance departments. Each of these might have its own allotment of storage, and indeed, in extreme cases, its own storage hierarchy. Fragment 9-2 illustrates a sequence of policy rules applied to files based on ownership.

Fragment 9-2 Using file ownership to control placement

```
<RULE Name=”Engineering_Rule”>
  <SELECT>
    <GROUP>Eng</GROUP>
  </SELECT>
    CREATE, RELOCATE, and DELETE statements for engineering
</RULE>

<RULE Name=”Admin_Rule”>
  <SELECT>
    <GROUP>Sales</GROUP>
    <GROUP>Mktg</GROUP>
    <GROUP>Service</GROUP>
  </SELECT>
    CREATE, RELOCATE, and DELETE statements for sales, marketing, & service
</RULE>

<RULE Name=”Finance_Rule”>
  <SELECT>
    <GROUP>Finance</GROUP>
  </SELECT>
    CREATE, RELOCATE, and DELETE statements for finance files
</RULE>
```

In Fragment 9-2, files owned by members of the `Eng` group are subject to `Engineering_Rule`, which might have one or more private placement classes or which might share some or all of its storage with files owned by other groups. The `Sales`, `Mktg`, and `Services` groups' files follow a common creation, relocation, and deletion rule, again, either with dedicated storage or with some or all of its placement classes shared with those specified in other rules. Finally, files owned by users in the `Finance` group are subject to yet a different rule.

File placement by logical position in the file system name space

The third selection criterion for file placement policy rules is the directory in which a file resides. Files can be selected by placement rules based on their logical locations in the file system name space. This facility can be used to control data location in a variety of ways. For example, new or patched applications may be placed on low-cost storage for testing against space-saving snapshots of live data. When test results are satisfactory, the test versions can be moved to production directories, and the superseded versions can be moved to archival directories on low-cost storage. Assuming that rules for unconditional relocation of files in production directories are part of the active policy, the next policy enforcement relocates the superseded application images to archival storage (from which they can be moved offline or deleted when a suitable confidence interval has elapsed) and relocated the new versions to the appropriate volumes for production applications.

Directory locations can also be used to achieve an effect similar to activity-based relocation. For example, in any transaction-oriented application there are records and documents whose access frequency is high when they are new and decreases as they age. If I/O activity decreases to zero, the access age relocation qualifier can be used. If I/O activity decreases but does not cease, average I/O temperature or average access temperature can be used as the relocation qualifier. In both cases, individual file locations reflect actual I/O activity against the files. Some aged files may be on top-tier storage, while others of the same age may have been relocated downward.

An alternative to these methods is to move files into alternate directories as they age, and define policy rules that cause files in those directories to be relocated to other storage tiers. Fragment 9-3 shows a policy rule in which logical location in the file system name space selects files for unconditional relocation to second-tier volumes.

Fragment 9-3 Using directory location to control file placement

```
[1]     <RULE Name="Current_Transactions_Rule">
[2]       <SELECT>
[3]         <DIRECTORY>current</DIRECTORY>
[4]       </SELECT>
[5]       <CREATE><ON Flags="any"><CLASS>top_tier</CLASS></ON></CREATE>
[6]     </RULE>

[7]     <RULE Name="Aged_Transactions_Rule">
[8]       <SELECT>
[9]         <DIRECTORY Flags="Recursive">aged</DIRECTORY>
[10]      </SELECT>
[11]      <RELOCATE><TO><CLASS>second_tier</CLASS></TO></RELOCATE>
[12]    </RULE>
```

The first rule in lines `[1]-[6]` of Fragment 9-3 places new files in the `top_tier` class, or in other placement classes if `top_tier` fills up. The second rule in lines `[7]-[12]` unconditionally relocates files that reside in the `/mnt/aged` directory and all of its subdirectories (as indicated by the `Flags="Recursive"` attribute) the `second_tier` placement class.

This policy might be appropriate for use with an operating procedure that created new transaction and document files each month in a `/mnt/current` directory, and, at month's end, moved all files in the directory to directories `/mnt/aged/jan`, `/mnt/aged/feb`, and so forth. The next time this policy is enforced, it relocates files in these directories to `second_tier` volumes without any further administrative action.

File placement by file size

DST policies can relocate files based on their sizes at the time of policy enforcement. This feature can be useful with files that start out small and grow during their lifetimes to the point where they could overwhelm the volumes on which they are originally allocated, preventing allocation of additional files.

Fragment 9-4 Size-based policy rules

```
<RELOCATE>
  <FROM><SOURCE><CLASS>small_file_tier</CLASS></SOURCE></FROM>
  <TO><DESTINATION><CLASS>large_file_tier</CLASS></DESTINATION></TO>
  <WHEN>
    <SIZE Units="MB"><MIN Flags="gt">100</MIN></SIZE>
  </WHEN>
</RELOCATE>

<RELOCATE>
  <FROM><SOURCE><CLASS>large_file_tier</CLASS></SOURCE></FROM>
  <TO><DESTINATION><CLASS>small_file_tier</CLASS></DESTINATION></TO>
  <WHEN>
    <SIZE Units="days"><MAX Flags="lteq">100</MAX></SIZE>
  </WHEN>
</RELOCATE>
```

Placing files to give preferential service to users

For service bureau-type data centers that cater to large numbers of external clients, Dynamic Storage Tiering can be used as a business differentiator. Using any of the aforementioned criteria, files that relate to individual users can be placed on specific storage tiers. Fragment 9-5 gives two examples of how DST can be used to provide different levels of service to different classes of users.

Fragment 9-5 Delivering different service levels to different classes of users

```
<RULE Name="Premium_User_Rule">
  <SELECT>
    <DIRECTORY Flags="Recursive">premium</DIRECTORY>
  </SELECT>
  <CREATE><ON Flags="any"><CLASS>top_tier</CLASS></ON></CREATE>
</RULE>

<RULE Name="Ordinary_User_Rule">
```

Fragment 9-5 Delivering different service levels to different classes of users *(continued)*

```
[8]       <SELECT>
[9]         <DIRECTORY Flags="Recursive">ordinary</DIRECTORY>
[10]      </SELECT>
[11]      <CREATE><ON Flags="any"><CLASS>second_tier</CLASS></ON></CREATE>
[12]    </RULE>

[13]    <RULE Name="Alt_Premium_User_Rule">
[14]      <SELECT>
[15]        <GROUP>premium_group</GROUP>
[16]      </SELECT>
[17]      <CREATE><ON Flags="any"><CLASS>top_tier</CLASS></ON></CREATE>
[18]    </RULE>

[19]    <RULE Name="Alt_Ordinary_User_Rule">
[20]      <SELECT>
[21]        <GROUP>ordinary_group</GROUP>
[22]      </SELECT>
[23]      <CREATE><ON Flags="any"><CLASS>second_tier</CLASS></ON></CREATE>
[24]    </RULE>
```

Lines `[1]-[12]` of Fragment 39 direct files anywhere in the `premium` directory subtree to the `top_tier`placement class and files in the `ordinary` subtree to the second-tier class. This would be useful, for example, if users were assigned to subdirectories in one or the other of these subtrees depending upon whether they paid for premium or ordinary service. The rules in lines `[13]-[24]` make the same assignments, but based on whether files' owners are in group `premium-group` or in group `ordinary_group`. These rules would be more appropriate for systems in which users controlled the directories in which they created files, but were assigned to one of the two groups based on the level of service for which they had contracted.

Fragment 9-5 illustrates file creation, but differential relocation and deletion services are possible as well. For example, ordinary users might have files deleted after 90 days of inactivity, while premium users' files are not subject to deletion. Premium users might be allowed 30 days of inactivity before their files were relocated downward, while ordinary users' files are subject to relocation after 7 days of inactivity. Size might also be used as a service discriminator. Premium users might be permitted to keep 100 megabyte files on `top_tier` volumes, whereas ordinary users' files might be relocated if they grew beyond 10 megabytes. The general principle of this example is that DST can be used to provide differential quality of service to different classes of users, provided that the users can be identified by one of the DST file selection criteria.

Chapter 10

Using Dynamic Storage Tiering: storage reclamation

This chapter includes the following topics:

Getting the benefit of multi-tier storage

The Dynamic Storage Tiering facility reduces online storage cost without offsetting increases in administrative cost because it keeps infrequently-used data easily accessible within a file system name space, while allowing it to be stored on less costly storage devices. In many instances, retrofitting Dynamic Storage Tiering to existing file systems and relocating inactive files to low-cost storage makes it possible to reclaim premium-quality storage capacity for other uses. If a premium volume is made up of disks whose capacity is not shared with other volumes, it can be shrunk, and the freed capacity can be made available for other purposes.

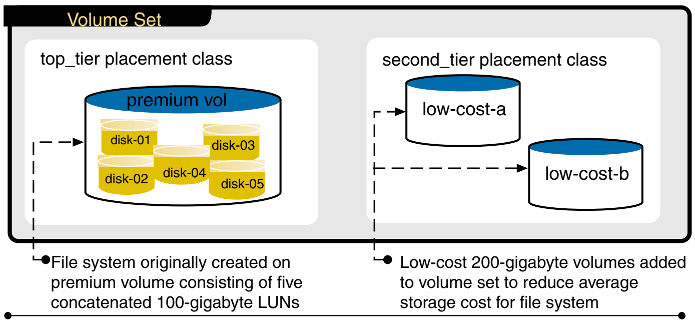
With Storage Foundation Version 5 installed, the `vxdg upgrade` command in line `[3]` upgrades the VxVM volume layout version to a level that is compatible with Dynamic Storage Tiering. The `mkfs` command with the `-m` option in line `[4]` lists the characteristics of an existing file system. This file system's layout is `version=6`, and must be upgraded to Version 7, which is accomplished by the command in line `[8]`.

Reclaiming and repurposing storage

A multi-volume file system's complement of storage can be expanded by adding volumes to any of its storage tiers. One common use for this capability is the addition of a second tier of one or more low-cost volumes to a file system that was originally created on a single premium-quality volume. Inactive, large, or infrequently accessed files are then relocated to the second-tier volumes, freeing space on the premium one and reducing the average cost of storage for the file system as a whole. Premium-quality storage made available by installing a second tier can be redeployed in new applications, deferring the need for future purchases of such storage.

But the savings from moving less critical data to lower-cost storage cannot be realized until the premium-quality storage capacity freed by relocating files has been "taken away" from the file system and deployed elsewhere. The sections that follow use a simple example to illustrate adding a second tier of low-cost storage to a file system, relocating files to it, and reclaiming the excess unused top-tier storage space. Figure 10-1 illustrates the file system's volume set configuration for this example before and after the addition of the second tier.

Figure 10-1 Two-tier volume set for storage reclamation example


Volume Set
top_tier placement class
premium vol
disk-01
disk-02
disk-03
disk-04
disk-05
second_tier placement class
low-cost-a
low-cost-b
File system originally created on premium volume consisting of five concatenated 100-gigabyte LUNs
Low-cost 200-gigabyte volumes added to volume set to reduce average storage cost for file system

Reclaiming storage: an example

Figure 10-1 represents the storage complement of a file system that was originally formatted on a single premium-quality volume, `premium-vol`, created from five 100-gigabyte LUNs presented by an enterprise-class disk array. If the LUNs themselves are mirrored or RAID-protected, the volume might be concatenated or striped by VxVM. Alternatively, the volume might be a VxVM-striped and mirrored aggregate of non-redundant LUNs.

A second tier of storage consisting of two 200-gigbyte volumes created from low-cost hardware devices is added to the file system's volume set. The enterprise's intention is to reduce the overall cost of storage used by this file system by retrofitting Dynamic Storage Tiering to it, relocating inactive files to the lower-cost second-tier volumes, and if possible, reclaiming part of the premium volume's capacity for deployment elsewhere.

Table 10-1 lists the steps required of an administrator to retrofit DST to an existing file system and use it to relocate inactive files to second-tier storage.

Table 10-1 Retrofitting Dynamic Storage Tiering facility to an existing file system

Step	Description
Create volume set and add volumes to it	A file system must reside on a VxVM volume set containing at least two volumes in order for a meaningful DST policy to be assigned to it.
Tag volumes	The tiers of storage presented as volumes in a file system's volume set must be tagged with the `vxfs.placement_class.tag_name` tags used in the placement policy. The examples in this chapter use `tier1` and `tier2` as the placement class names.
Define and assign file placement policy	The file placement policy specifies initial placement and relocation of files to second-tier and other volumes based on relevant business criteria such as file inactivity, size, average I/O temperature, or average access temperature.
Enforce policy	Enforcing the policy for the first time relocates lower-value data to newly-added storage tiers. Later enforcements ensure that premium-quality storage is reserved for files deemed critical according to the active placement policy.

The Dynamic Storage Tiering facility requires that file systems to which it is applied be formatted with VxFS layout Version 7 or a newer one, and that volumes use VxVM layout Version 140 or a newer one. Dialog 10-1 illustrates a sequence of commands that determine the volume layout version of the disk group containing the file system's volume set and upgrade it, as well as ascertaining the file system layout and upgrading it to Version 7.

Dialog 10-1 Upgrading VxVM and VxFS layout versions

```
# vxdg list dstdg | grep version: | awk '{ print $2 }'
105
# vxdg upgrade dstdg

# mkfs -F vxfs -m /dev/vx/dsk/dstdg/premium-vol
mkfs -F vxfs -o bsize=1024,version=6,inosize=256,logsize=1024,largefiles
/dev/vx/dsk/dstdg/premium-vol
```

Dialog 10-1 Upgrading VxVM and VxFS layout versions *(continued)*

```
524288
# vxupgrade -n 7 /dst
```

Preparing for multi-tier storage

Dialog 10-2 shows the steps that prepare for retrofitting Dynamic Storage Tiering to the file system for this example.

Dialog 10-2 Preparing tier2 storage for a multi-volume file system

```
[1]      # umount /dst
[2]      # vxvset -g dstdg make dstvset premium-vol

[3]      # mount -F vxfs /dev/vx/dsk/dstdg/dstvset /dst
[4]      # vxassist -g dstdg make low-cost-a 200G

[5]      # vxassist -g dstdg make low-cost-b 200G
[6]      # vxvset -g dstdg addvol dstvset low-cost-a

[7]      # vxvset -g dstdg addvol dstvset low-cost-b
[8]      # fsvoladm add /dst low-cost-b 419430400

[9]      # fsvoladm add /dst low-cost-b 419430400
[10]     # vxvoladm -g dstdg settag premium-vol vxfs.placement_class.tier1
[11]     # vxvoladm -g dstdg settag low-cost-a vxfs.placement_class.tier2
[12]     # vxvoladm -g dstdg settag low-cost-b vxfs.placement_class.tier2
```

The `vxvset make` command in line `[2]` creates the `dstvset` volume set from the `premium-vol` volume. Conversion of a single volume into a volume set leaves data on the volume intact. The `vxvset addvol` commands in lines `[4]-[7]` create two low-cost volumes and add them to the `dstvset` volume set. The `fsvoladm add` commands in lines `[8]-[9]` make the low-cost volumes accessible to the file system. Finally, the commands in lines `[10]-[12]` tag the volumes, effectively creating the `tier1` and `tier2` placement classes.

The purpose of retrofitting DST is to relocate less-critical files to low-cost volumes, freeing space on the premium volumes so that one of them can be reclaimed and used elsewhere. Less-critical files may be identified by inactivity, low temperature, directory, naming pattern, owner or owner's group, or any combination of these. Fragment 10-1 shows a simple policy that classifies data for relocation to `tier2` volumes based on a combination of naming pattern and inactivity.

Fragment 10-1 Policy for supplanting top-tier storage with lower-cost second-tier storage

```
[1]     <?xml version="1.0"?>
[2]     <!DOCTYPE PLACEMENT_POLICY SYSTEM "/opt/VRTSfspro/config/placement_policy.dtd">
[3]     <PLACEMENT_POLICY Version="5.0" Name="accmod">

[4]       <RULE Flags="data" Name="db_files">
[5]         <SELECT><PATTERN>*.db</PATTERN></SELECT>
[6]         <CREATE><ON>
[7]           <DESTINATION><CLASS>tier1</CLASS></DESTINATION>
[8]         </ON></CREATE>
[9]         <RELOCATE>
[10]          <TO><DESTINATION><CLASS>tier2</CLASS></DESTINATION></TO>
[11]          <WHEN><MODAGE Units="days"><MIN Flags="gteq">7</MIN></MODAGE></WHEN>
[12]        </RELOCATE>
[13]      </RULE>

[14]      <RULE Flags="data" Name="other_files">
[15]        <SELECT><PATTERN>*</PATTERN></SELECT>
[16]        <CREATE><ON>
[17]          <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
[18]        </ON></CREATE>
[19]        <RELOCATE>
[20]          <TO><DESTINATION><CLASS>tier2</CLASS></DESTINATION></TO>
[21]        </RELOCATE>
[22]      </RULE>
[23]    </PLACEMENT_POLICY>
```

The placement policy in Fragment 10-1 treats database files, identified by the naming pattern `*.db`, differently from all other files. Database files are created on `tier1` volumes, and relocated to `tier2` volumes if they are inactive for seven or more days. Once relocated to `tier2` volumes, they are never relocated back to `tier1`. Because the file system initially occupies only `tier1` volumes, there are no active database files on `tier2` volumes if the policy is assigned to the file system immediately after the `tier2` volumes are added to its volume set.

The rule in lines `[14]-[22]` of Fragment 10-1 is the catchall rule that covers all files other than database ones. New files that are not named according to the `*.db` pattern are allocated on `tier2` volumes. Existing files that reside on `tier1` volumes are relocated to `tier2` unconditionally whenever the policy is enforced.

Enforcing a placement policy to reduce premium storage occupancy

Lines `[1]-[6]` of Dialog 10-3 illustrate the assignment of the policy to the file system by an authorized administrator and the resulting distribution of data immediately after assignment as well as after the passage of time.

Dialog 10-3 Space distribution prior to policy enforcement

```
[1]     # /opt/VRTSfspro/bin/fsppadm assign /dst /tmp/accmod.xml
[2]     # fsvoladm list /dst
[3]     devid   size                 used                 avail                 name
[4]     0       524288000            344288000            180000000            premium-vol
[5]     1       209715200            16                   209715184             low-cost-a
[6]     2       209715200            16                   209715184             low-cost-b

           ... applications use file system...


[7]     # fsvoladm list /dst
[8]     devid   size                 used                 avail                 name
[9]     0       524288000            344289120            179998880            premium-vol
[10]    1       209715200             61002040            148713160            low-cost-a
[11]    2       209715200             80998000            128717200            low-cost-b
```

As lines `[5]-[6]` of Dialog 10-3 suggest, the low-cost `tier2` volumes do not contain any data immediately after they are added to the file system's volume set. (They are not completely empty because VxFS uses the first the first 16 kilobytes of each volume for internal purposes.)

Lines `[7]-[11]` of Dialog 10-3 show the state of volume occupancy after the file system has been used for some time, but before the active policy has been enforced. Newly-allocated non-database files occupy space on the `tier2` volumes, but because there has been no enforcement, neither inactive database files nor non-database files that existed prior to policy assignment have been relocated.

Dialog 10-4 shows the results of a `fsppadm query` command issued by an authorized administrator to determine the probable impact of a policy enforcement.

Dialog 10-4 Prediction of the effect of policy enforcement

```
[1]     # fsppadm query /dst
[2]     Current     Current     Relocated   Relocated
[3]     Class       Volume      Class       Volume      Rule        File
[4]     tier1       premium-vol tier2       low-cost-a  db_files    /dst/hist-oct.db
[5]     tier1       premium-vol tier2       low-cost-a  db_files    /dst/hist-nov.db

           ...additional database file relocations...


[6]     tier1       premium-vol tier2       low-cost-b  other_files /dst/temp/file_a
[7]     tier1       premium-vol tier2       low-cost-b  other_files /dst/temp/file_b


           ...additional non-database file relocations...
```

Dialog 10-4 Prediction of the effect of policy enforcement *(continued)*

```
[8]      tier1       premium-vol                        db_files    /dst/shares.db
[9]      tier1       premium-vol                        db_files    /dst/employees.db

            ...additional database files not to be relocated...
```

The results of the `fsppadm query` command in lines `[2]-[9]` of Dialog 10-3 indicate that inactive database files would be relocated because of the `db_files` rule, and non-database files would be relocated because of the `other_files` rule, if the policy were to be enforced. Active database files (lines `[8]-[9]`) would be left in place, again due to the `db_files` rule. The `fsppadm query` command does not relocate the files; it provides a preview of what relocations would occur if the active policy were to be enforced. It is primarily useful for determining what would be the effect of enforcing the active policy without actually incurring the overhead of file movement during enforcement.

Dialog 10-5 illustrates an ad hoc policy enforcement command issued by an authorized administrator, and the resulting distribution of allocated space across the file system's volume set.

Dialog 10-5 Policy enforcement and resulting allocated space distribution

```
[1]      # fsppadm enforce /dst
[2]      Sweep path     : /dst
[3]      Files moved    : 2398
[4]      KB moved       : 200000000

[5]      Tier Name             Size (KB)  Free Before (KB)   Free After (KB)
[6]      tier2                 419430400         277430360         142000040
[7]      tier1                 524288000         179998880         379998880

[8]      #  fsvoladm list /dst
[9]      devid   size                 used                 avail                name
[10]     0       524288000            144289120            379998880           premium-vol
[12]     1       209715200            141002040             68713160           low-cost-a
[13]     2       209715200            200998000              8717200          low-cost-b
```

The space distribution in Dialog 10-5 reflects the relocation of database files that have been inactive for seven or more days prior to the time of enforcement, as well as all non-database files, whether active or not, from `tier1` to `tier2` volumes. Policy enforcement reacts to conditions in the file system at the time of enforcement. If a file system's active policy is enforced immediately following a `fsppadm query`, the results will closely resemble the prediction from the query. If, however, time elapses between the two, results may differ, depending on activity in the file system between the two events.

A file placement policy can be enforced at any time. If, however, the policy contains any <IOTEMP> or <ACCESSTEMP> relocation qualifiers, no deletions or relocations based on those qualifiers will occur unless the File Change Log is running, `filestats` storage is enabled, and the amount of data saved in the FCL represents a period at least as large as the largest <PERIOD> specified in the policy.

Reclaiming storage capacity released by file relocation

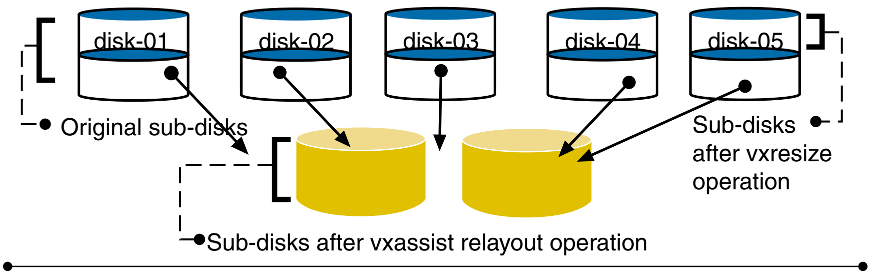
With occupancy of the `premium-vol` volume reduced to about 142 gigabytes, volume's size can be reduced, and the space reclaimed for other purposes using the command sequence shown in Dialog 10-6.

Dialog 10-6 Reducing file system and volume size

```
[1]     # vxresize -F vxfs -g dstdg premium-vol -300G
[2]     # fsvoladm list /dst
[3]     devid   size                used                avail               name
[4]     0       209715200           144289120            79998880            premium-vol
[5]     1       209715200           141002040            68713160           low-cost-a
[6]     2       209715200           200998000             8717200           low-cost-b
[7]     # vxassist -g dstdg relayout premium-vol layout=stripe ncol=2
```

The `vxresize` command in Dialog 10-6 reduces the size of the `premium-vol` and the por-tion of the file system that resides on it from 500 gigabytes to 200 gigabytes. If the vol-ume is a concatenated one, VxVM removes the sub-disks on three of the 100 gigabyte LUNs that comprise it from the volume, and returns them to the disk group's pool, where they are available for other uses. If the volume is a striped one, the `vxresize` step shrinks each of the sub-disks by an equal amount, leaving the freed space distributed as the top half of Figure 10-2 illustrates.

Figure 10-2 Results of resize and relayout operations


disk-01
disk-02
disk-03
disk-04
disk-05
Original sub-disks
Sub-disks after vxresize operation
Sub-disks after vxassist relayout operation

If `premium-vol` is striped, the space released by the `vxresize` operation must be consolidated onto two of the sub-disks. The `vxassist relayout` command in line `[7]` of Dialog 10-6 accomplishes this, leaving the `premium-vol` disk configuration as illustrated in the bottom half of Figure 10-2 and returning the three freed disks to the `dstdg` disk group's pool of available storage.

When reclaiming storage from a volume on which a single-volume file system was originally formatted, care must be taken to ensure that sufficient `metadataok` storage capacity remains available to the file system to meet anticipated future needs. If a file system runs out of `metadataok` storage, no files can be allocated or extended, even though there may be free capacity on `dataonly` volumes in its volume set. By default, the first volume in a file system's volume set (`premium-vol` in this example) is `metadataok`, and all other volumes are `dataonly`. Changing the eligibility of a volume in a volume set to contain file system metadata is discussed on Dialog 5-2.

Conclusion

For file systems created using Version 5 of the Storage Foundation, administrators have the luxury of provisioning the estimated storage required in each tier in the form of volumes that can easily be re-provisioned for other purposes by simply evacuating them and removing them from the file system's volume set. For file systems that were created with earlier versions of the Storage Foundation, however, Dynamic Storage Tiering must usually be retrofitted by converting the file system's volume to a volume set, adding additional storage tiers and volumes, and assigning a policy that relocates files across the expanded volume set according to business requirements. While this procedure can be followed with any type of storage, the most frequent application is to reduce the overall cost of storage for a large file system by adding second-tier storage to it and relocating inactive or otherwise-qualified files to it. The storage capacity of the original volume that is released by doing this can be consolidated by evacuating specified sub-disks, after which the volume can be shrunk by removing the evacuated sub-disks from it. Sub-disks reclaimed in this way can be repurposed, for example, by creating new volumes and file systems on them, or by adding them to the volume sets of other file systems that require additional storage of the type they represent. If the sub-disks represent physical disks or LUNs presented by a disk array, they can be deleted, and the disks or LUNS removed from their disk group and deployed on other hosts.

Chapter 11

Using Dynamic Storage Tiering: managing storage for databases

This chapter includes the following topics:

Databases and file systems

Reclamation of unneeded storage is the “rest of the story” of Dynamic Storage Tiering. It makes application of DST to existing file systems practical because it eliminates the need for massive data copy operations or lengthy service outages to retrofit the facility and reclaim storage capacity that is freed by using it.

Most commercial databases store their data in files. The file systems they use are unusual in that they typically contain relatively small numbers of large files with long lifetimes. Particularly for large databases, it is crucial that each file reside on the appropriate type of storage throughout its lifetime. The Dynamic Storage Tiering facility offers unique capabilities for controlling placement of database component files within a multi-volume file system. DST file placement policies can place database files appropriately when they are created, and relocate them as conditions change throughout their lifetimes.

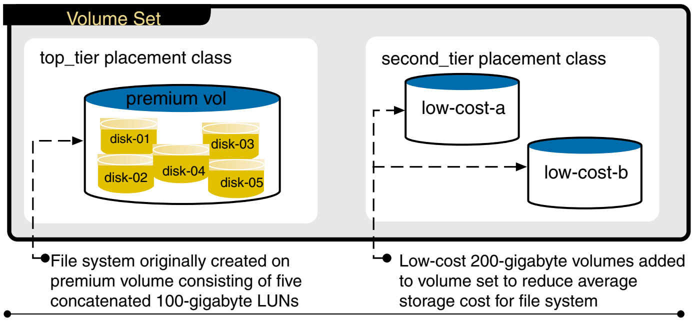
The DBDST facility

The Database Editions of the Storage Foundation implement special functions to manage database file placement for the market-leading database management systems—DB2, Oracle, and Sybase. Collectively, these functions are called Database Dynamic Storage Tiering (DBDST) facilities. DBDST functions are accessed through commands that are designed to be natural for database administrators to use. Database Edition software uses the capabilities of the Dynamic Storage Tiering facility to control the placement of database files within a multi-volume file system.

The DBDST facility simplifies the management actions that database administrators must take to manage multi-tier storage in a database environment. The sections that follow illustrate the capabilities of the DBDST facility by showing how it can be used to solve common database storage management problems.

Preparing database storage to use the DBDST facility

Figure 11-1 Database storage configuration for use with the DBDST facility


Volume Set
top_tier placement class
premium vol
disk-01
disk-03
disk-04
disk-02
disk-05
second_tier placement class
low-cost-a
low-cost-b
File system originally created on premium volume consisting of five concatenated 100-gigabyte LUNs
Low-cost 200-gigabyte volumes added to volume set to reduce average storage cost for file system

To make use of the DBDST facility, a database must store its data files in a VxFS multi-volume file system whose volumes are tagged with placement class names that are used in file placement policies created by the database administrator. Figure 11-1 illustrates a volume configuration for a database originally created using a file system (mounted at `/oradata`) on a single volume (`oradata`). The file system's volume configuration is augmented by the addition of both high-performance and low-cost volumes so that the DST facility can be used to manage file placement within the file system.

By default, DBDST registers the placement class names `PRIMARY`, `SECONDARY`, and `BALANCE` for each database file system registered with it. These class names may optionally be used when tagging volumes and creating placement policies, or other class names may be created and used. A database administrator uses the `dbdst_admin addclass` command to register additional placement class names with DBDST, as Dialog 11-1 illustrates.

Dialog 11-1 Adding volume tag names to the DBDST registry

```
[1]     # dbdst_admin -S proddb -o addclass "tier1:Fast Storage for Production DB"
[2]     # dbdst_admin -S proddb -o addclass "tier2:Slow Storage for Production DB"
```

The commands in Dialog 11-1 register the placement class names `tier1` and `tier2` with DBDST for the file system that contains the `proddb` database. Once registered, these names can be used in DBDST file placement policy definitions for the `proddb` database.

Converting database file systems

In the example of Figure 11-1, in order to use DBDST capabilities, the `/oradata` file system must be converted into a multi-volume file system with at least two volumes in its volume set, one of which is its original volume (`oradata`). The `dbdst_convert` command illustrated in Dialog 11-2 creates a volume set based on the original `oradata` volume, adds volumes `new_vol1`, `new_vol2`, and `new_vol3` to it, and converts the `/oradata` file system into a multi-volume one.

Dialog 11-2 Converting to a multi-volume file system with multiple volumes

```
[1]     # dbdst_convert -S proddb -g oradg -M /dev/vx/dsk/oradg/oradata \
[2]     new_vol1,new_vol2,new_vol3
```

VxVM does not permit a volume and a volume set to have the same name. During conversion, DBDST renames the original volume by appending the string `_b4vset` to its original name. The original volume name becomes the volume set name. In the example of Figure 11-1, the oradata volume is renamed as `oradata_b4vset`, and the new volume set is named `oradata`. Because volume sets behave as volumes for administrative purposes, this permits any pre-existing storage administration commands and scripts that operate on volumes to execute unaltered.

The single `dbdst_convert` command in Dialog 11-2 creates the volume set configuration shown in Figure 11-1 and converts the `/oradata` file system to a multi-volume one, effectively by issuing a series of Storage Foundation commands similar to that contained in Dialog 4-2. DBDST encapsulates several Storage Foundation management operations into this single command. Moreover, using

DBDST, a database administrator can perform the conversion without requiring access to facilities normally restricted to system administrators.

Putting volumes in placement classes

When a database's file system has been converted, and additional volumes have been added, registered placement classes can be assigned to individual volumes using `dbdst_classify` commands similar to those illustrated in Dialog 11-3.

Dialog 11-3 Assigning placement classes to volumes

```
[1]    # dbdst_classify -S proddb -g oradg  -M /dev/vx/dsk/oradg/oradata \
       -v oradata_b4vset:tier1
[2]    # dbdst_classify -S proddb -g oradg  -M /dev/vx/dsk/oradg/oradata -v new_vol1:tier1
[3]    # dbdst_classify -S proddb -g oradg  -M /dev/vx/dsk/oradg/oradata -v new_vol2:tier2
[4]    # dbdst_classify -S proddb -g oradg  -M /dev/vx/dsk/oradg/oradata -v new_vol3:tier2
```

The commands in Dialog 11-3 assign the tag `tier1` to volumes `oradata_b4vset` and `new_vol1`, and the tag `tier2` to volumes `new_vol2` and `new_vol3`. These tags correspond to placement classes that are used in file placement policies.

With the execution of these commands, the example database's storage configuration is as shown in Figure 11-1. This configuration, with the `/oradata` multi-volume file system, and two placement classes is ready for DST policy assignment and enforcement. The sections that follow describe how DST capabilities are invoked through the Database Dynamic Storage Tiering (DBDST) facility to solve several common database storage management problems.

Problem 1: infrequent relocation of Oracle tablespaces and partitions

Perhaps the simplest application of multi-tier storage to databases is relocation of individual table partitions between different placement classes as usage requirements change. If exact relocation times are unpredictable, or if relocation is infrequent, administrators may wish to relocate table partitions as business requirements surface rather than defining strict periodic relocation schedules.

Ad hoc relocation of table partitions can be useful, for example, with databases that track sales and inventory for seasonal businesses such as sports equipment or outdoor furniture retailing. As the selling season for one type of inventory (for example, summer equipment or furniture) approaches, database table partitions that represent in-season goods can be relocated to high-performance storage, since they will be accessed frequently during the coming months. Similarly, partitions that represent out-of-season goods can be relocated to lower-cost storage, since activity against them is likely to be infrequent.

For example, a table in an inventory database might be divided into `winter` and `summer` partitions, with each of these partitions mapped to a separate data file. As the winter selling season approaches, the administrator for this database might issue commands similar to those in Dialog 11-4.

Dialog 11-4 On-demand relocation of database table partitions (part 1)

```
[1]     # dbdst_partition_move -S proddb -T product_table -p winter -c tier1
[2]     # dbdst_partition_move -S proddb -T product_table -p summer -c tier2
```

The commands in Dialog 11-4 relocate the files that comprise the `winter` partition of the `product_tab` table to placement class `tier1`, and the files that comprise the `summer` partition to placement class `tier2` (assuming a volume set configuration similar to that illustrated in Figure 11-1). DBDST determines which files comprise the `winter` and `summer` partitions of `product_table`, and uses underlying DST services to immediately relocate those files to the `tier1` and `tier2` placement classes respectively.

A similar pair of commands, illustrated in Dialog 11-5 can be executed at the onset of the summer selling season to reverse the respective positions of the partitions.

Dialog 11-5 On-demand relocation of database table partitions (part 2)

```
[1]     # dbdst_partition_move -S proddb -T product_table -p winter -c tier2
[2]     # dbdst_partition_move -S proddb -T product_table -p summer -c tier1
```

DBDST formulates DST policy rules that unconditionally relocate the files containing the target partitions to the destination placement classes. It merges these rules into the database file system's active policy, assigns the resulting composite policy to the file system, and enforces it immediately to relocate the subject files. Because the added policy rules precede any other rules in the active policy, the subject files remain in place until the `dbdst_partition_move` command is next executed, at which time the rules are removed and replaced with others.

A similar DBDST command, `dbdst_tbs_move`, illustrated in Dialog 11-6, relocates entire tablespaces to alternate placement classes.

Dialog 11-6 On-demand relocation of an entire table

```
[1]     # dbdst_tbs_move -S proddb -t customer_tbs -c tier1
```

The command in Dialog 11-6 relocates the files that comprise the `customer_tbs` tablespace in the `proddb` database to `tier1` volumes when the policy is enforced. DBDST identifies the files that comprise the `customer_tbs` tablespace and adjusts the file system's DST policy accordingly.

Unscheduled on-demand relocation of Oracle database partitions and entire tables is useful for databases with seasonal changes or those with variations in activity that are not precisely predictable, as described in this example. Issuing the commands when relocation is appropriate automatically adjusts for variations in time-based events that necessitate relocation of partitions or tables. On-demand partition and table relocation is also useful for dealing with one-time events, such as table addition and re-partitioning, as well as changes in business circumstances that change database table or partition performance and availability requirements.

Problem 2: identifying changes in activity level and relocating files

One frequently-encountered business circumstances that affects data placement is age-based change in activity level against one or more database files. This is common, for example, in transaction-processing systems and systems that support customer service or relationship management applications. Files containing tables that represent current transactions tend to be very active as new transactions are added and clients inquire about recent activities. As records age, activity against them tends to decrease, to the point where they become essentially archival in nature.

For applications like billing, activity level as a function of time is quite predictable. This type of data lends itself to regularly scheduled relocation of files that contain aged transaction records to lower-cost storage tiers. For other applications, such as customer relationship management, changes in activity against database files may not be so easy to predict. To support the latter, database manager statistics collection tools can be used, or alternatively, the Database Dynamic Storage Tiering facility can collect statistics on database file I/O activity. A database administrator can use the `dbdst_fstat start` command to enable collection of I/O statistics on specified lists of files, as line `[1]` of Dialog 11-7 illustrates.

Dialog 11-7 Collecting and using statistics on database file I/O activity

```
[1]     # dbdst_fstat -S proddb -o start -f dbfile_list

        ... database runs with statistics collection active ...

[2]     # dbdst_file_move -S proddb -f inactive_dbfile_list -c tier2
[3]     # dbdst_file_move -S proddb -f active_dbfile_list -c tier1
```

The command in line `[1]` of Dialog 11-7 starts collection of I/O activity statistics against database files whose names are listed in the file `dbfile_list`, which is created by the database administrator. When DBDST statistics collection is active, DBDST queries the file system with a default frequency of 30 minutes (changeable by the administrator), and stores the results against the listed files in its own repository.

After I/O statistics collection has been active for sufficient time to discern meaningful activity patterns, database performance tools can be queried to determine the database‘s most and least active files. Alternatively, the DBDST graphical console can display a list of the most and least active files. From either of these sources, an administrator can create a file listing the most or least used of the database files on which statistics are being collected (`inactive_file_list` and `active_file_list` in Dialog 11-7), which can be input to the `dbdst_file_move` command to relocate the files as required.

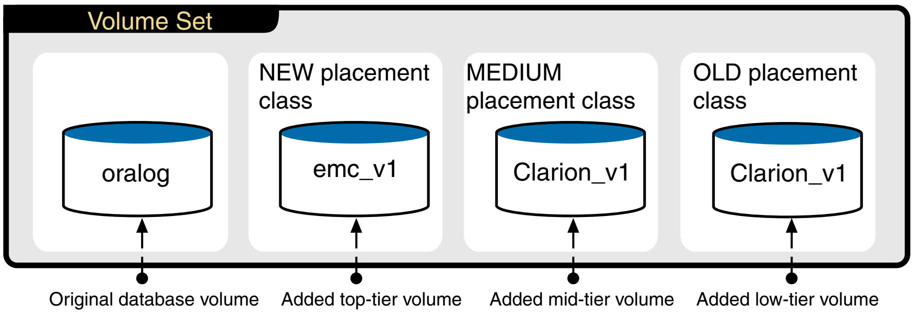
Problem 3: scheduled relocation of archive and flashback logs

Because they are the primary mechanism for recovering from data corruption, database logs are normally kept on premium storage, both for I/O performance and data reliability reasons. Even after they have been archived, logs are normally kept online for fast recovery, but the likelihood of referring to an archived log decreases significantly as its age increases. This suggests that archived database logs might be relocated to lower-cost volumes after a certain period of inactivity.

Similarly, Storage Foundation Database Edition Flashback technology creates logs that can be used for quick recovery from database corruption by restoring a database to its state at a previous time. Flashback logs are normally kept for a shorter period than archived database logs—if used at all, they are typically used within a few hours of creation. Two or three days is a typical Flashback log lifetime.

The rapidly decaying probability of use for archive and Flashback logs suggests that regular enforcement of a placement policy that relocates them to lower-cost storage after a period of inactivity can reduce an enterprise’s average cost of online storage. Figure 11-2 illustrates a three-tier volume configuration that is suitable for automatic relocation and deletion of archive logs and Flashback logs.

Figure 11-2 Database storage configuration for archived and Flashback log relocation



The file system used by the production database in this example originally resides on the single volume `oralog`; it must be prepared as in the preceding example, additional volumes added to it, and placement classes assigned to its volumes. Dialog 11-8 illustrates these preparatory steps.

Dialog 11-8 Preparing for automatic relocation of archived and database logs

```
[1]      # dbdst_admin  -S proddb -o addclass "NEW:EMC Storage for Production DB"
[2]      # dbdst_admin  -S proddb -o addclass "MEDIUM:Clarion Storage for Production DB"
[3]      # dbdst_admin  -S proddb -o addclass "OLD:JBOD Storage for Production DB"

[4]      # dbdst_convert -S proddb -g oradg -M /dev/vx/dsk/oradg/oralog \
[5]      -v emc_v1,clarion_v1,jbod_v1

[6]      # dbdst_classify -S proddb -g oradg  -M /dev/vx/dsk/oradg/oralog  emc_v1:NEW
[7]      # dbdst_classify -S proddb -g oradg  -M /dev/vx/dsk/oradg/oralog  clarion_v1:MEDIUM
[8]      # dbdst_classify -S proddb -g oradg  -M /dev/vx/dsk/oradg/oralog  jbod_v1:OLD
```

Lines `[1]-[3]` of Dialog 11-8 register the `NEW`, `MEDIUM`, and `OLD` placement classes in the DBDST repository. Lines `[4]-[5]` convert the `proddb` database's file system and add volumes `emc_v1`, `clarion_v1` and `jbod_v1` to it. Lines `[6]-[8]` assign placement classes (volume tags) `NEW`, `MEDIUM`, and `OLD` to the new volumes.

Once the volumes are configured for DST use, an administrator can define file placement policy rules that specify access age-based relocation of selected files and assign them to the database's file system. Dialog 11-9 illustrates the use of the `dbdst_file_move` command to define a rule that relocates files in the `flashback` and `archive1` directories to lower tiers in the new storage hierarchy.

Dialog 11-9 Defining policy rules that periodically relocate flashback and archive logs

```
[1]  # dbdst_file_move -S proddb -o flashback -c MEDIUM:2
[2]  # dbdst_file_move -S proddb -o archive1 -c MEDIUM:7 -c OLD:15
```

The command in line `[1]` of Dialog 11-9 relocates files in the `flashback` directory which have not been accessed for two days to the `MEDIUM` volume. The command in line `[2]` relocates files in the `archive1` directory that have not been accessed for seven days to the `MEDIUM` volume, and files that have not been accessed for 15 days to the `OLD` volume.

DBDST translates these commands into DST access age-based policy rules, merges them with the file system's placement policy, and assigns the resulting policy to the file system. By default, DBDST enforces the active policy daily. During enforcement, the new rules relocate qualifying files to the destination storage tiers specified in the commands in Dialog 11-9. Commands similar to these can be used to relocate archive database logs and flashback logs for both Oracle and DB2 databases.

Problem 4: relocating external files

Many databases contain metadata about large numbers of external files, such as images, documents, experimental data, and so forth. Because external files tend to be large, it can be advantageous to relocate inactive ones to lower tiers in the storage hierarchy. If the external files reside in a known set of directories, a DBA can use DBDST to create policy rules that relocate inactive ones during daily placement policy enforcement. The `dbdst_file_move` command creates policy rules for a list of directories specified in a file such as that shown in Dialog 11-10.

Dialog 11-10 Creating policy rule that relocates files from a list of directories

```
[1]      # cat external_file_dirs
[2]      /proddb/pictures/Alabama
[3]      /proddb/pictures/California
[4]      /proddb/binary/proprietary
         ^c

[5]      # dbdst_file_move -S proddb -o external -f external_file_dirs -c MEDIUM:7 -c OLD:30
```

Assuming the volume set configuration illustrated in Figure 11-2, the `dbdst_file_move` command in line `[5]` of Dialog 11-10 creates a DST policy rule that relocates external files based on their access ages. The command results in augmentation of the database file system's active placement policy rules with a rule that selects files in the directories listed in the `external_file_dirs` file, relocates those that have been inactive for seven days to the `MEDIUM` volume, and relocates those that have been inactive for 30 days to the `OLD` volume. This form of access age-based external file relocation can be used with both Oracle and DB2 databases.

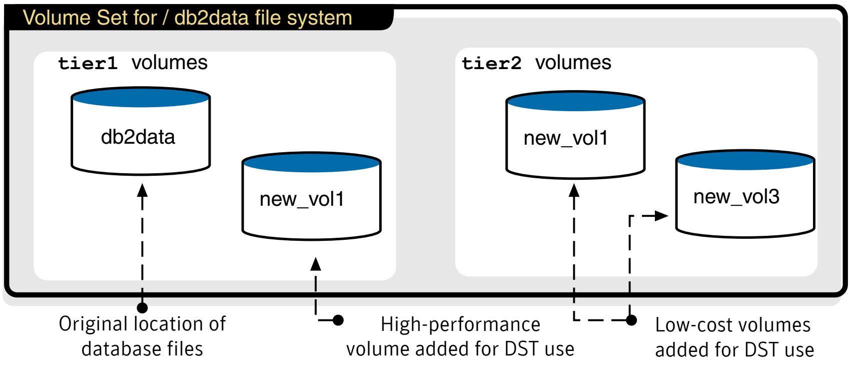
Problem 5: preset placement policies for database files

In many cases it is desirable to permanently segregate certain types of database files from other data managed by a system for performance or availability reasons. For example, a data center might choose to reserve its highest-quality storage for database datafiles and index files, and relegate other data to a second tier in the storage hierarchy.

DBDST preset file placement policies make such segregation possible. An administrator can create a preset placement policy and assign it to a database's file system as the active DST policy.

DBDST preset policies are useful with DB2's System Managed Storage (SMS) feature. When SMS is in use, DB2 creates files as required by database operations within the confines of a "tablespace container," a designated directory in the database's file system. Figure 11-3 shows the storage configuration for a file system that contains files for the DB2 database `db2db` as well as other files. The enterprise wishes to reserve `tier1` volumes for DB2 data and index files, and allocate all other files on `tier2` volumes.

Figure 11-3 Database storage configuration for DB2 SMS example


Volume Set for / db2data file system
tier1 volumes
db2data
new_vol1
tier2 volumes
new_vol1
new_vol3
Original location of database files
High-performance volume added for DST use
Low-cost volumes added for DST use

Using the `dbdst_preset_policy` command, an administrator can create an SMS directory, and create and assign a file placement policy, as Dialog 11-11 illustrates.

Dialog 11-11 Creating a SMS container and a preset file placement policy

```
# dbdst_preset_policy -D db2db -o sms -d /db2data/tbs01 \
-P tier1=*.DAT,*.INX
```

The command in Dialog 11-11 creates the SMS tablespace container directory `/db2data/tbs01`, as well as a file placement policy similar to that shown in Fragment 11-1 for the file system `/db2data`.

Fragment 11-1 Segregating database files from other files

```
[1]     <RULE Name="DB2 Files Rule">
[2]       <SELECT>
[3]         <DIRECTORY Flags="recursive">tbs01</DIRECTORY>
[4]         <PATTERN>*.DAT</PATTERN>
[5]         <PATTERN>*.INX</PATTERN>
[6]       </SELECT>
[7]       <CREATE><ON><DESTINATION><CLASS>tier1</CLASS></DESTINATION></ON></CREATE>
[8]       <RELOCATE><TO><DESTINATION><CLASS>tier1</CLASS></DESTINATION></TO></RELOCATE>
[9]     </RULE>

[10]    <RULE Name="Other Files Rule">
[11]      <SELECT><PATTERN>*</PATTERN></SELECT>
[12]      <CREATE><ON><DESTINATION><CLASS>tier2</CLASS></DESTINATION></ON></CREATE>
[13]    </RULE>
```

The policy rule in lines `[1]-[8]` of Fragment 11-1 places files in directory `tbs01` and its subdirectories whose names follow either of the patterns `*.DAT` or `*.INX` on `tier1` volumes. The rule in lines `[9]-[12]` places all other files on `tier2` volumes.

Once the preset policy has been created and assigned to the database's file system, a database administrator can use DB2 administrative commands to create the SMS tablespace `tbs01` specifying `/db2data/tbs01` as the tablespace container. Space for data and index files created by DB2 during the course of operations are allocated on `tier1` volumes; all other files are allocated on `tier2` volumes. DBDST also enforces the policy to relocate already-existing database files to the appropriate placement class.

If the DB2 SMS container directory exists, the form of the `dbdst_preset_policy` command shown in Dialog 11-12 can be used to create, assign, and enforce a policy of the form shown in Fragment 11-1.

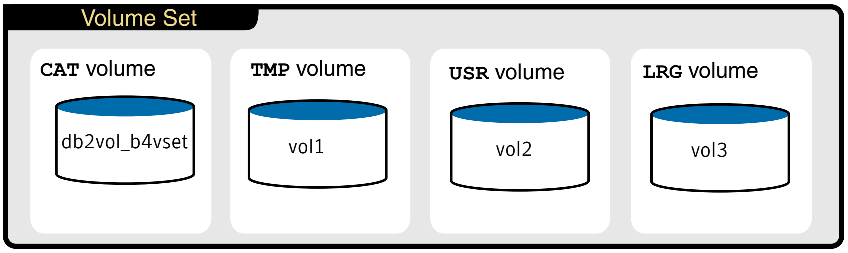
Dialog 11-12 Creating a preset file placement policy for an existing a SMS container

```
# dbdst_preset_policy -D db2db -o sms -t tbs01 -P tier1=*.DAT,*.INX
```

Using DBDST with DB2 automatic storage path

DBDST can also be used to place different types of data in DB2 databases that use DB2's automatic storage path feature on different placement classes. Figure 11-4 illustrates one typical scheme for placing different types of database data on different storage tiers.

Figure 11-4 Database storage configuration for separating different types of DB2 data



Dialog 11-13 shows the DBDST commands used to register these placement classes. In each case, the comment field indicates the types of data to be stored in the class.

Dialog 11-13 Placement class definitions for separating different types of DB2 data

```
[1]     # dbdst_admin  -D db2db -o addclass CAT:"system catalog tablespace"
[2]     # dbdst_admin  -D db2db -o addclass TMP:"system or user temporary tablespace"
[3]     # dbdst_admin  -D db2db -o addclass USR:"user or regular tablespace"
[4]     # dbdst_admin  -D db2db -o addclass LRG:"large tablespace"
```

The database in this example is originally stored in a file system on the `db2vol` volume, the command in line `[1]` of Dialog 11-14 converts the hosting file system and adds three additional volumes, `vol1`, `vol2`, and `vol3`, to it.

Dialog 11-14 Assigning placement classes to volumes

```
[1]     # dbdst_convert  -D db2db -M /dev/vx/dsk/db2dg/db2vol -v vol1,vol2,vol3
[2]     # dbdst_classify -D db2db -M /dev/vx/dsk/db2dg/db2vol -v db2vol_b4vset:CAT
[3]     # dbdst_classify -D db2db -M /dev/vx/dsk/db2dg/db2vol -v vol1:TMP
[4]     # dbdst_classify -D db2db -M /dev/vx/dsk/db2dg/db2vol -v vol2:USR
[5]     # dbdst_classify -D db2db -M /dev/vx/dsk/db2dg/db2vol -v vol3:LRG

[6]     dbdst_preset_policy -D db2db -o ats -P CAT=*.CAT:TMP=*.TMP:USR=*.USR:LRG=*.LRG
```

The commands in lines `[2]-[5]` assign placement classes to the four volumes, resulting in the configuration shown in Figure 11-4.

The automatic storage path feature of DB2 requires a directory, which for this example is `/db2db/ats`. The `dbdst_preset_policy` command creates and assigns a policy similar to the one shown in Fragment 11-2 that selects files in this directory and creates them in the appropriate placement classes.

Fragment 11-2 Placing different database files on different placement classes

```
          <RULE Name="CAT Files Rule">
[1]         <SELECT>
[2]           <DIRECTORY Flags="recursive">ats</DIRECTORY>
[4]           <PATTERN>*.CAT</PATTERN>
[6]         </SELECT>
[7]         <CREATE><ON><DESTINATION><CLASS>CAT</CLASS></DESTINATION></ON></CREATE>
[8]         <RELOCATE><TO><DESTINATION><CLASS>CAT</CLASS></DESTINATION></TO></RELOCATE>
[9]       </RULE>

[10]      <RULE Name="TMP Files Rule">
[11]        <SELECT>
[12]          <DIRECTORY Flags="recursive">ats</DIRECTORY>
[13]          <PATTERN>*.TMP</PATTERN>
[14]        </SELECT>
[15]        <CREATE><ON><DESTINATION><CLASS>TMP</CLASS></DESTINATION></ON></CREATE>
[16]        <RELOCATE><TO><DESTINATION><CLASS>TMP</CLASS></DESTINATION></TO></RELOCATE>
[17]      </RULE>

[18]      <RULE Name="USR Files Rule">
[19]        <SELECT>
[20]          <DIRECTORY Flags="recursive">ats</DIRECTORY>
[21]          <PATTERN> *.USR</PATTERN>
[22]        </SELECT>
[23]        <CREATE><ON><DESTINATION><CLASS>USR</CLASS></DESTINATION></ON></CREATE>
[24]        <RELOCATE><TO><DESTINATION><CLASS>USR</CLASS></DESTINATION></TO></RELOCATE>
[25]      </RULE>

[26]      <RULE Name="LRG Files Rule">
[27]        <SELECT>
[28]          <DIRECTORY Flags="recursive">ats</DIRECTORY>
[29]          <PATTERN> *.LRG</PATTERN>
[30]        </SELECT>
[31]        <CREATE><ON><DESTINATION><CLASS>LRG</CLASS></DESTINATION></ON></CREATE>
[32]        <RELOCATE><TO><DESTINATION><CLASS>LRG</CLASS></DESTINATION></TO></RELOCATE>
[33]      </RULE>

[34]      <RULE Name="Other Files Rule">
[35]        <SELECT><PATTERN>*<PATTERN></SELECT>
[36]        <CREATE><ON><DESTINATION><CLASS>LRG</CLASS></DESTINATION></ON></CREATE>
[37]      </RULE>
```

Each of the first four rules in Fragment 11-2 (lines `[1]-[8]`, lines `[9]-[15]`, lines `[16]-[22]`, and lines `[23]-[29]`) selects one of the target file types based on naming pattern and creates them in the desired placement class. The fifth rule, in lines `[31]-[34]`, is the catchall rule that selects all other files and places them on the `LRG` volume. Without this rule, allocation of any files not selected by the first four rules would fail, so omitting it would be a way of limiting the types of files that can be created in the `/db2db` file system.

Once this policy has been created and assigned, the DB2 database can be created, setting `/db2db/ats` as the automatic storage path. Again, by using DBDST facilities, the database administrator avoids the necessity of coordinating with system administration each time a database storage configuration is created or changes.

Although Oracle does not require specific name patterns for its data files, pattern-based placement policies similar to those described in this section can be adopted for Oracle databases as well, provided that database administrators follow file naming conventions. For example, Oracle datafiles are named according to the pattern `*.dbf`. Indexes are typically placed in separate tablespaces, in files named according to pattern `*.inx`. If these naming conventions are adhered to, it is possible to separate datafiles and indexes from other files in the file system using the `dbdst_preset_policy` command as described in this section to create a preset file placement policy similar to that shown in Fragment 11-2.

Problem 6: increasing performance by load balancing

A heavily loaded database can exceed the I/O performance limits of its storage devices. In some cases, the load can be distributed by segregating different files or different types of files on different volumes. In other cases, this solution is not possible, for example, because the load is due to accesses to a single table, partition, or datafile.

When an I/O load saturates the capacity of the devices to which it is directed, the most common solution is to distribute the load across more resources—across more disks and I/O paths. DBDST includes a `dbdst_makelbfs` command that creates a multi-volume file system and assigns it a DST policy containing the `<BALANCE_SIZE>` sub-element that distributes the allocation of database files across the volumes in its volume set. In principle, distributing the data in each file across multiple volumes distributes the I/O load across them as well. Dialog 11-15 illustrates the use of the `dbdst_make_lbfs` command to create the load-balancing file system `/customer_projects`.

Dialog 11-15 Using DBDST to create a load-balancing file system

```
[1]   # dbdst_make_lbfs -M /customer_projects -g proddg -C 1M -v vol1,vol2,vol3,vol4
```

The command in Dialog 11-15 creates the file system on volumes `vol1`, `vol2`, `vol3`, and `vol4` (all of which are placed in the `BALANCE` placement class by DBDST). In addition, it creates and assigns a file placement policy that distributes files across the four volumes with a `<BALANCE_SIZE>` of 1 megabyte. (For Oracle databases, the balance size must be 1 megabyte.) When the database manager allocates or extends files, VxFS distributes the allocated space randomly across the volumes in the `BALANCE` placement class in extents of 1 megabyte. This has the effect of randomly distributing the I/O load approximately evenly across the four volumes.

If the I/O capacity of a load-balancing file system proves inadequate, additional volumes can be added to the file system using the `dbdst_addvol` command, as Dialog 11-16 illustrates.

Dialog 11-16 Adding a volume to a DBDST load-balancing file system

```
[1]   # dbdst_addvol -M /dev/vx/dsk/proddg/vol1 -v newvol:BALANCE
```

The command in Dialog 11-16 adds volume `newvol` to the volume set of file system `/customer_projects`, and to placement class `BALANCE`. After the volume is added, VxFS balances file allocations and extensions in the `/customer_projects` file system across all five volumes in the `BALANCE` placement class.

Chapter 12

Using Dynamic Storage Tiering: exploiting MAID-based storage

This chapter includes the following topics:

The right storage for the right data

Chapter 2 presents the rationale for the multi-tier storage concept. In essence, today's storage hardware and configuration technology present users with a range of I/O performance and data availability options at different price points. Thus it is possible for enterprises to choose storage for their data that is commensurate with the data's value to the business.

Storage configuration options range from simple or striped (non-failure-tolerant) disk drives, through RAID LUNs instantiated by disk arrays, to multiply-mirrored LUNs that are remotely replicated for disaster recovery. Almost any configuration option can be implemented using a range of hardware technologies, from low-cost disk drives embedded in the server enclosure, through mid-range disk arrays, to enterprise disk arrays with high-performance components and advanced failure tolerance and I/O performance features. The multi-tier storage premise is that for every data set an enterprise stores online, there is a "right" type of storage from an I/O performance, data availability, and cost point of view.

Why multi-tier storage works

The multi-tier storage concept “works” because of a simple fact of information technology life—much of the data that enterprises store online is archival in nature, and therefore seldom accessed by applications. Such data changes seldom, if at all, and only occasionally, if ever, does access by applications make I/O performance an issue. The obvious conclusion is that seldom-accessed data can be stored on low-cost storage devices at considerable savings to the enterprise storage budget over time.

The preponderance of occasionally accessed data that must nevertheless be kept online for operational or other business reasons has doubtless had a strong influence on the introduction of disk arrays that utilize desktop disk drives to provide low-cost storage for data that is not accessed frequently. Today, most major storage vendors offer low-cost alternatives to their enterprise-class products, and promote them specifically as storage for seldom-accessed data.

MAID: A new tier in the storage hierarchy

More recently, vendors have begun to offer even more specialized disk array-like products for seldom accessed data that are called Massive Arrays of Idle Disks (MAID). The underlying premise of MAID-based storage is simple: if most of the data stored on the array is seldom accessed, the disks that hold it need not be powered on and spinning most of the time. If most of the disks in an array are powered off at any given moment, then the array’s power supplies, cooling devices, and disk controllers can all be sized for the small fraction of disks that are powered on and working at any instant. The result is greater packaging density, lower power consumption, higher disk reliability, and lower cost, both for acquisition and for operation. Table 12-1 describes each of these characteristics in more detail.

Table 12-1 MAID storage system characteristics

Storage characteristic	Description
Power and cooling	Because only a fraction of the disks in a typical MAID storage system are powered on at any instant (typically 25%), power supplies and cooling devices can be configured for roughly a quarter of the cost and volume of those in a conventional disk array in which all disks are spinning all the time.

Table 12-1 MAID storage system characteristics *(continued)*

Storage characteristic	Description
Packaging density	A MAID storage system occupies a considerably smaller footprint in the data center than a conventional disk array of similar capacity for two reasons. First, there are fewer power supplies, cooling devices, and disk controllers than in a conventional array. Second, because not all disks are powered on at the same time, they can be packaged more densely than disks in conventional arrays without fear of head damage. A MAID system typically stores about four times as many gigabytes per square foot as a conventional disk array.
Disk reliability	MAID storage systems utilize low-cost disk drives because the expected duty cycle is low. By powering disk drives down when they are not in use, a MAID system tends to extend disk drive life beyond manufacturer's design expectations.
Acquisition cost	As disk prices continue to fall, more of the cost of a disk array lies in the enclosure, power and cooling components, and control logic. MAID storage systems use fewer of these components per disk drive than conventional disk arrays, typically by factors of 4-8, thus making them less expensive per byte than conventional disk arrays that use low-cost drives.
Operating cost	MAID storage systems have a lower operating cost per online gigabyte than conventional storage devices simply because they consume less power (for disk drives and for cooling fans) and floor space.

The MAID value proposition, then, is online storage for occasionally accessed data at very low acquisition and operating cost compared to conventional disk arrays. Typically, the MAID cost differential leads to lifetime savings of 60-75% per online gigabyte.

Limitations of MAID-based storage

The very factor that makes MAID-based storage so cost-effective, power-managed disks, limits the applications for which it is suitable. When an application requests data on a disk that is not powered on, the disk must spin up to operating speed before the request can be satisfied, a matter of seconds. Moreover, because MAID systems are configured so that only a fraction (typically 25%) of their disks can be powered on at any one time, other disks may have to be powered off before a request can be satisfied. Thus, MAID-based storage is most suitable for applications for which a time-to-first-byte of a few seconds is tolerable. Moreover, MAID-based storage should be matched to applications in such a way that it is unlikely that more than the number of disks that can be powered on simultaneously in a single system will be accessed concurrently.

Applications for MAID-based storage

Like conventional disk array systems, MAID systems virtualize the LUNs they present to hosts. In so doing, they provide disk failure tolerance (typically RAID-based), and moreover, mask much of the complexity inherent in powering the right disks up and down at the right times. Once powered on, disks in a MAID-based system perform like equivalent disks in a conventional array. Disks are powered off when they are idle or when I/O requests directed at other disks in the array require that power be directed elsewhere. This suggests that the ideal application for MAID-based storage is for large data sets that are accessed occasionally, and when accessed, are read or written in their entirety. A number of applications have this characteristic. Table 12-2 lists some of the more common ones.

Table 12-2 Suitable applications for MAID-based storage

Application	Description
Seismic exploration	Large streams of raw exploration data are typically stored and read in their entirety by applications that reduce and analyze them. Resulting data sets are stored as streams, and when accessed for further processing, are accessed in their entirety as well.
High-energy physics research	Large streams of raw or preprocessed data from experiments are stored and read in their entirety by applications that reduce and enhance them with auxiliary information. Enhanced data streams are stored, and when accessed for analysis, are read in their entirety.
Financial analysis	Streams of raw financial data (e.g., ticker tape data) are stored and read in their entirety by applications that enrich them with auxiliary information and store the resulting enriched streams. The enriched streams are read in their entirety by applications that analyze the data in them to deduce trends on which decisions can be based.
Entertainment	Audio-visual segments are created and stored as streams. Streams are read in their entirety by applications that ultimately create streams representing finished pieces. Finished piece streams are stored, and retrieved for enhancements such as sound effects and language translation. Individual objects are also retrieved for reuse in creating other finished pieces.

All four of these applications use storage in similar ways. All read and write large streams of data. Any given stream is likely to be read infrequently (daily or less).

Applications may hold the data in a stream locally during processing, but ultimately they rewrite the original stream or create new streams based on the originals. The new streams may be read and processed locally by additional applications, but again, the ultimate result is yet another new stream that is stored permanently and accessed occasionally.

Dynamic Storage Tiering and MAID-based storage

One additional characteristic of the data streaming applications described in the preceding section is that their usage of data streams is ad hoc. When an application will read or write a permanent stream cannot be predicted based on the stream's access or modification age, on historical average I/O temperature or access temperature, or on the size of the stream. From a Dynamic Storage Tiering point of view, this means that the `<WHEN>` clause in the `RELOCATE` statement is of little utility in relocating streams of data between volumes based LUNs presented by MAID storage systems and volumes based on conventional storage devices. DST can, however, be used in conjunction with some simple operating procedures to manage the locations of these large data streams in a multi-volume file system whose volume set consists partly of volumes based on conventional (always-on) storage devices and partly of MAID-based volumes. Figure 12-1 illustrates a sample of such a procedure.

Figure 12-1 Operating procedures for use with MAID and conventional storage

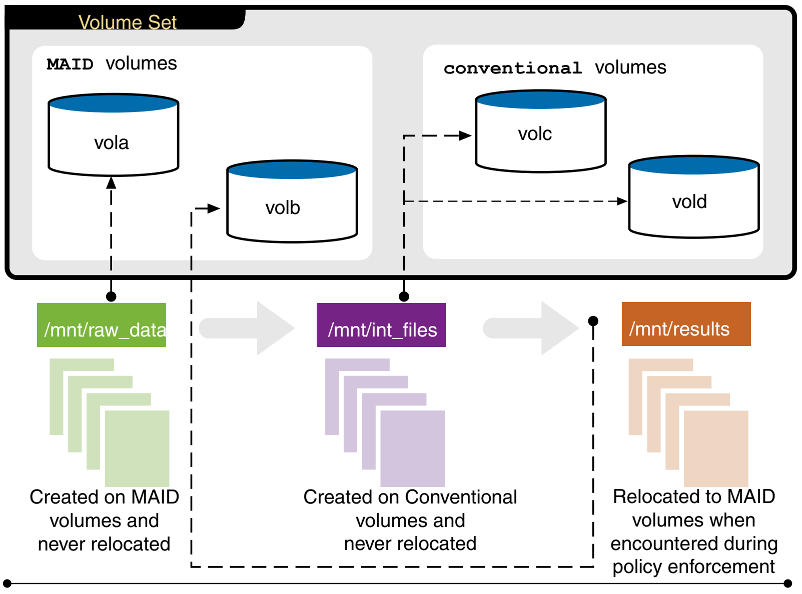

Volume Set
MAID volumes
vola
volb
conventional volumes
volc
vold
/mnt/raw_data
/mnt/int_files
/mnt/results
Created on MAID volumes and never relocated
Created on Conventional volumes and never relocated
Relocated to MAID volumes when encountered during policy enforcement

Figure 12-1 illustrates an operating procedure that might be typical of any of the applications listed in Fragment 12-1. Raw data (e.g., seismic traces, ticker data, experimental results, etc.) that is to be retained long term is stored in the `MAID` placement class. These volumes would be created using LUNs presented by a MAID storage system.

Applications would read the raw data directly from the `MAID` volumes, and use `Conventional` volumes to store and manipulate temporary intermediate files. Presumably, the `Conventional` placement class would be based on conventional disk array LUNs, and provide the performance typical of that type of device.

When producing permanent result files (e.g., final videos, enriched exploration or experimental data, etc.), applications would create the files on Conventional volumes, again for performance reasons. The Dynamic Storage Tiering facility can relocate these files to `MAID`-based volumes when the policy is enforced.

Policies for using MAID-based storage

Fragment 12-1 lists policy rules that support the mode of operation in which MAID-based volumes are used for long-term storage and conventional volumes are used for intermediate processing.

Fragment 12-1 Policy rules for utilizing MAID-based volumes

```
<RULE Name=”Raw Data Rule”>
   <SELECT><DIRECTORY Flags="recursive">raw_data</DIRECTORY></SELECT>
   <CREATE><ON><DESTINATION><CLASS>MAID</CLASS></DESTINATION></ON></CREATE>
 </RULE>

 <RULE Name=”Intermediate Files Rule”>
   <SELECT><DIRECTORY Flags="recursive">int_files</DIRECTORY></SELECT>
   <CREATE><ON>
     <DESTINATION><CLASS>Conventional</CLASS></DESTINATION>
   </ON></CREATE>
 </RULE>

 <RULE Name=”Results Rule”>
   <SELECT><DIRECTORY Flags="recursive">results</DIRECTORY></SELECT>
   <CREATE><ON>
     <DESTINATION><CLASS>Conventional</CLASS></DESTINATION>
   </ON></CREATE>
   <RELOCATE><DESTINATION><CLASS>MAID</CLASS></DESTINATION></RELOCATE>
 </RULE>

 <RULE Name=”All Other Files Rule”>
   <SELECT><PATTERN>*</PATTERN></SELECT>
   <CREATE><ON>
     <DESTINATION><CLASS>Conventional</CLASS></DESTINATION>
   </ON></CREATE>
 </RULE>
```

The policy rules in Fragment 12-1 are predicated on the observation of certain standard operating procedures. For example, when raw data files are created from input streams, they must be created in the directory `/mnt/raw_data` or in a sub-directory. According to the rule in lines `[1]-[4]`, these files are created on `MAID`-class volumes, and, because the rule contains no `RELOCATE` statement, are never relocated.

Similarly, the second rule in lines [5]-[10] is based on the assumption that intermediate files created by applications from raw data will be created in `/mnt/int_files` or one of its sub-directories. Space for files in these directories is allocated on `Conventional`-class volumes, and the files are never relocated. Presumably, these files would be short-lived, and so would not occupy expensive conventional storage for long periods.

The third rule in lines `[11]-[17]` deals with result files that are to be retained indefinitely. Long-term result files are assumed to be created in `/mnt/results` or in one of its sub-directories. These files are created on `Conventional`-class volumes, and relocated to `MAID`-class volumes when the policy is enforced. Applications can either create their permanent result files directly in `/mnt/results`, or they can create them elsewhere and move them into `/mnt/results` when they are complete and ready to be relocated to `MAID`-class volumes for long-term storage.

Other policy considerations

What is unique about this type of policy is that the single `RELOCATE` statement contains no qualifiers. Applications like those listed in Table 12-2 typically run at irregular intervals, determined primarily by human interaction. The concepts of access and modification age average I/O temperature, and average access temperature are of little relevance. Raw data files that are created `MAID` volumes should stay there indefinitely, and result files created on `Conventional` volumes should be relocated to `MAID` volumes as soon as the policy is enforced. The function that drives this policy is the directories in which files reside.

An additional policy consideration for MAID-based storage is enforcement planning. There are two important and interrelated planning considerations, defining MAID-based volumes and scheduling policy enforcement. MAID-based volumes and the schedules on which they are used should be defined carefully because of the limited number of disks in a MAID storage system that can be powered on simultaneously. Because MAID systems provide internal failure tolerance, volumes concatenated from LUNs presented by the array are usually optimal, because when a volume is accessed, only the LUN currently being accessed need be powered on. As a corollary to this, the ideal number of volumes configured is equal to the number of LUNs that can be powered on simultaneously. For example, if a MAID system presents 32 LUNs, and can support 8 powered-on simultaneously, the ideal volume configuration would be 8 volumes, with each consisting of four concatenated LUNs. If 8 applications write or read data separate to or from separate volumes simultaneously, each volume would have one LUN active at a time (the one reading or writing data), thus remaining within the array's power budget.

Part III

Appendixes

Appendix A

Sample XML for a built-in file placement policy

This appendix includes the following topics:

- Output of the built-in policy creation wizards
- Summary: common properties of built-in file placement policies

Output of the built-in policy creation wizards

This appendix contains a representation of a placement policy document produced by the built-in placement policy wizard example in Chapter 6. Some of the XML text has been reformatted for readability, but the entire text of the policy is represented.

The preferred files rule: file creation

The access aged-based with preferred files policy described from the VEA graphical wizard standpoint in Chapter 6 generates four rules—one for preferred files, one each for exempt files pinned to the top and bottom storage tiers, and one for all other files. The four appear in that order in the resulting policy. Fragment A-1, in addition to the XML header lines, illustrates the `SELECT` and `CREATE` statements for the `Preferred-Files-Rule`.

Fragment A-1 Preferred files rule—SELECT and CREATE statements

```
<?xml version="1.0"?>
<!DOCTYPE PLACEMENT_POLICY SYSTEM "/opt/VRTSfspro/config/placement_policy.dtd">
<PLACEMENT_POLICY Name="Database-Policy" Version="5.0" Guid="{dfaa5748-1dd1-
11b2-a73d-0003ba44a638}" Generation="1">
  <RULE Name="Preferred-Files-Rule" >
    <SELECT>
```

Fragment A-1 Preferred files rule—SELECT and CREATE statements *(continued)*

```
[6]         <USER>dbadmin</USER>
[7]         <USER>chiefdba</USER>
[8]         <USER>hrdba</USER>
[9]         <PATTERN>*.db</PATTERN>
[10]        <DIRECTORY>database</DIRECTORY>
[11]      </SELECT>

[12]      <CREATE><ON>
[13]          <DESTINATION><CLASS>Tp</CLASS></DESTINATION>
[14]          <DESTINATION><CLASS>T2</CLASS></DESTINATION>
[15]          <DESTINATION><CLASS>T3</CLASS></DESTINATION>
[16]        </ON></CREATE>
```

To be a preferred file selected by this rule, a file must be owned by one of users `dbadmin`, `chiefdba`, or `hrdba`, must be in the `database` directory (relative to the file system mount point), and in addition, must have a name that follows the pattern `*.db`. Meeting only one or two of these conditions is insufficient.

The `<DESTINATION>` clauses of the `CREATE` statement in Fragment A-1 are a priority-ordered list. VxFS allocates space for newly-created preferred files on the `Tp` volume if possible, resorting to the `T2` and `T3` volumes (in that order) if the `Tp` volume has insufficient free space. New preferred files are not allocated on `T1` volumes unless all `Tp`, `T2`, and `T3` volumes are fully occupied, causing VxFS to choose locations for new files. (Preferred files are never relocated to the `T1` volume.)

The preferred files rule: downward file relocation

Fragment A-2 illustrates the two `RELOCATE` statements that control the relocation of inactive preferred files from higher storage tiers to lower ones. If a built-in policy specifies N placement classes, the wizard creates N-1 `RELOCATE` statements.

In keeping with the general principle of placing less inclusive statements ahead of more inclusive ones in a placement policy, the `RELOCATE` statement for files inactive for 20 or more days occurs before the statement for files inactive for 10 or more days. The first `RELOCATE` statement (lines `[1]-[8]`) specifies that files that reside on either the `Tp` volume or one of the `T2` volumes and that are inactive for 20 days or more should be relocated to one of the `T3` volumes. The second statement (lines `[9]-[16]`) specifies that files that reside on the `Tp` volume (only) and that are inactive for 10 days or more should be relocated to one of the `T2` volumes if space permits, and to one of the `T3` volumes if not. In other words, files that have been inactive for between 10 and 20 days are not relocated downward unless they reside on the `Tp` volume.

Fragment A-2 Preferred files rule—downward relocation statements

```
    <RELOCATE>
      <FROM>
        <SOURCE><CLASS>Tp</CLASS></SOURCE>
        <SOURCE><CLASS>T2</CLASS></SOURCE>
      </FROM>
      <TO><DESTINATION><CLASS>T3</CLASS></DESTINATION></TO>
      <WHEN><ACCAGE Units="days"><MIN Flags="gt">20</MIN></ACCAGE></WHEN>
    </RELOCATE>

    <RELOCATE>
      <FROM><SOURCE><CLASS>Tp</CLASS></SOURCE></FROM>
      <TO>
        <DESTINATION><CLASS>T2</CLASS></DESTINATION>
        <DESTINATION><CLASS>T3</CLASS></DESTINATION>
      </TO>
      <WHEN><ACCAGE Units="days"><MIN Flags="gt">10</MIN></ACCAGE></WHEN>
    </RELOCATE>
```

The preferred files rule: upward file relocation

Fragment A-3 shows the companion upward `RELOCATE` statements for preferred files. Files are relocated from lower-tier volumes to higher-tier ones based on average I/O temperature, calculated using both bytes read and bytes written.

Fragment A-3 Preferred files rule—upward relocation statements

```
    <RELOCATE>
      <FROM>
        <SOURCE><CLASS>T2</CLASS></SOURCE>
        <SOURCE><CLASS>T3</CLASS></SOURCE>
      </FROM>
      <TO>
        <DESTINATION><CLASS>Tp</CLASS></DESTINATION>
        <DESTINATION><CLASS>T2</CLASS></DESTINATION>
      </TO>
      <WHEN>
        <IOTEMP Type="nrwbytes "><MIN Flags="gt">10</MIN><PERIOD>5</PERIOD></IOTEMP>
      </WHEN>
    </RELOCATE>

    <RELOCATE>
      <FROM><SOURCE><CLASS>T3</CLASS></SOURCE></FROM>
      <TO><DESTINATION><CLASS>T2</CLASS></DESTINATION></TO>
      <WHEN>
        <IOTEMP Type="nrwbytes "><MIN Flags="gt">5</MIN><PERIOD>5</PERIOD></IOTEMP>
      </WHEN>
    </RELOCATE>

  </RULE>
```

The first RELOCATE statement in Fragment A-3 (lines [1]-[13]) specifies that preferred files that reside on T2 and T3 volumes and whose five-day average I/O temperatures are greater than 10 should be relocated to the Tp volume if space permits, and to one of the T2 volumes if not. VxFS does not relocate files to volumes whose placement classes appear below those of the files' current classes in a list of <DESTINATION> elements. This can happen if four or more tiers are specified in one of the built-in policies.

The second RELOCATE statement (lines [14]-[21]) specifies that files residing on T3 volumes whose five-day average I/O temperatures are higher than 5 should be relocated to T2 volumes. As with downward relocation (Fragment A-2), this sequence of statements demonstrates the principle that less inclusive specifications should appear before more inclusive ones in a policy rule. In this sequence, more active files (those with average I/O temperatures greater than 90) from either T2 or T3 volumes are relocated first, to the Tp volume if possible, whereas slightly less active files (those with average I/O temperatures above 80 but less than 90) are only relocated if they reside on T3 volumes, and are only relocated to T2 volumes.

Rules for files exempted from relocation

Fragment A-4 shows the two rules that govern files that are exempt from relocation. The first of these rules (lines [1]-[15]]) applies to files that reside in the etc directory (relative to the file system mount point) or its subdirectories and are either named special-file.dat or have a name extension equal to conf following the period. These files are created on the T1 volume if space permits, and on the T2 and T3 volumes respectively if not. The RELOCATE statement (lines [12]-[14]) is unconditional, as indicated by the absence of both FROM and WHEN clauses—it specifies relocation of any of the designated exempt files that reside elsewhere to the T1 volume. This statement deals with situations in which space for exempt files has been allocated on other volumes due to lack of free space on T1 volumes, followed by T1 space becoming available at a later time.

Fragment A-4 Rules for exempt files pinned to top and bottom-tier volumes

```
[1]     <RULE Name="Key-Files-Rule" >
[2]       <SELECT>
[3]         <PATTERN>*.conf</PATTERN>
[4]         <PATTERN>special-file.dat</PATTERN>
[5]         <DIRECTORY Flags="recursive">etc</DIRECTORY>
[6]       </SELECT>
[7]       <CREATE><ON>
[8]           <DESTINATION><CLASS>T1</CLASS></DESTINATION>
[9]           <DESTINATION><CLASS>T2</CLASS></DESTINATION>
[10]          <DESTINATION><CLASS>T3</CLASS></DESTINATION>
[11]        </ON></CREATE>
```

Fragment A-4 Rules for exempt files pinned to top and bottom-tier volumes *(continued)*

```
[12]      <RELOCATE>
[13]        <TO><DESTINATION><CLASS>T1</CLASS></DESTINATION></TO>
[14]      </RELOCATE>
[15]    </RULE>

[16]    <RULE Name="Low-Files-Rule" >
[17]      <SELECT>
[18]        <DIRECTORY Flags=”nonrecursive”>tmp</DIRECTORY>
[19]      </SELECT>
[20]      <CREATE><ON>
[21]          <DESTINATION><CLASS>T3</CLASS></DESTINATION>
[22]          <DESTINATION><CLASS>T2</CLASS></DESTINATION>
[23]          <DESTINATION><CLASS>T1</CLASS></DESTINATION>
[24]        </ON></CREATE>
[25]      <RELOCATE>
[26]        <TO><DESTINATION><CLASS>T3</CLASS></DESTINATION></TO>
[27]      </RELOCATE>
[28]    </RULE>
```

The second rule (lines `[16]-[28]`), dealing with exempt files that are to be pinned to the lowest tier in the hierarchy (`T3`), specifies that files in the `tmp` directory (but not its subdirectories) are to be allocated on `T3` volumes if possible, and on `T2` volumes or the `T1` volume respectively only if free space is not available on any `T3` volume. Again, the `RELOCATE` statement (lines `[25]-[27]`) specifies unconditional relocation of qualifying files that reside elsewhere to `T3` volumes. This statement deals with situations in which files have been allocated on higher-tier volumes due to lack of free space on `T3` volumes, followed by free `T3` space becoming available at a later time.

Other files rule: file creation

Fragment A-5 shows the `SELECT` and `CREATE` statements in the policy rule that governs non-preferred files (all files not covered by `Preferred-Files-Rule`, `Key-Files-Rule`, and `Low-Files-Rule`). Because it occurs last, only files that are not designated by the preceding rules are evaluated against it. It is a “catchall” rule; its `<PATTERN>*</PATTERN>` clause selects all files that have not been selected by one of the three preceding rules. This rule specifies space allocation on the `T1` volume if possible, with overflow to the `T2` and `T3` volumes if necessary. Non-preferred files are only allocated on the `Tp` volume if no free space is available in the enumerated placement classes, causing VxFS to choose locations for new files.

Fragment A-5 Non-preferred, non-exempt file rule—SELECT and CREATE statements

```
[1]     <RULE Name="Normal-Files-Rule" >
[2]       <SELECT><PATTERN>*</PATTERN></SELECT>
[3]       <CREATE><ON Flags=”any”>
[4]           <DESTINATION><CLASS>T1</CLASS></DESTINATION>
[5]           <DESTINATION><CLASS>T2</CLASS></DESTINATION>
[6]           <DESTINATION><CLASS>T3</CLASS></DESTINATION>
[7]         </ON></CREATE>
```

Other files rule: file relocation

Fragment A-6 shows the `RELOCATE` statements for relocating non-preferred files from higher tiers in the storage hierarchy to lower ones. As with all built-in policies, specifying a hierarchy of N placement classes results in N-1 `RELOCATE` statements for both downward and upward relocation.

Fragment A-6 Non-preferred, non-exempt file rule—downward relocation statements

```
[1]       <RELOCATE>
[2]         <FROM>
[3]           <SOURCE><CLASS>T1</CLASS></SOURCE>
[4]           <SOURCE><CLASS>T2</CLASS></SOURCE>
[5]         </FROM>
[6]         <TO><DESTINATION><CLASS>T3</CLASS></DESTINATION></TO>
[7]         <WHEN><ACCAGE Units="days"><MIN Flags="gt">20</MIN></ACCAGE></WHEN>
[8]       </RELOCATE>

[9]       <RELOCATE>
[10]        <FROM><SOURCE><CLASS>T1</CLASS></SOURCE></FROM>
[11]        <TO>
[12]          <DESTINATION><CLASS>T2</CLASS></DESTINATION>
[13]          <DESTINATION><CLASS>T3</CLASS></DESTINATION>
[14]        </TO>
[15]        <WHEN><ACCAGE Units="days"><MIN Flags="gt">10</MIN></ACCAGE></WHEN>
[16]      </RELOCATE>
```

The first `RELOCATE` statement in Fragment A-6 (lines `[1]-[8]`) specifies that non-preferred files that reside on the `T1` volume or on one of the `T2` volumes should be relocated to one of the `T3` volumes if they have not been accessed in 20 or more days, and from the `T1` volume (only) to one of the `T2` volumes if they have not been accessed for 10 or more days (with `T3` volumes for overflow in this case). If no `T2` or `T3` space is available, no relocation occurs until the policy is enforced at a time when lower-tier space is available.

The 20 and 10 day inactivity periods are the same as those specified for preferred files (Fragment A-2). They are based on the input specifications given in the wizard page shown in Figure 6-7. This is a simplifying property of the built-in policies, not a limitation of the placement policy grammar. An administrator can use an XML or text editor to create policies that contain more general rules and statements. The attractions of the built-in policies are that they cover common business cases for effective multi-tier storage utilization, that they are very simple to create, and that they produce grammatically correct policy documents.

Fragment A-7 illustrates the final two rules in the access age-based with preferred files policy. These rules govern the relocation of non-preferred files located on other than the `T1` volume to higher tiers in the storage hierarchy. The first statement (lines `[1]-[13]`) specifies relocation of files on `T2` and `T3` volumes whose five-day average I/O temperatures are 90 or more to the `T1` volume. The second statement (lines `[14]-[24]`) specifies relocation of files on `T3` volumes whose five-day average I/O temperatures are 80 or more to `T2` volumes. These qualifiers are identical to those for relocating preferred files (Fragment A-3). Using the same average I/O temperature parameters as with classes of files is a simplifying property of the built-in policy mechanisms rather than a limitation of the placement policy grammar. Fragment A-7 also shows the XML closing delimiters for the `Normal-Files-Rule` and for the placement policy as a whole.

Fragment A-7 Non-preferred, non-exempt file rule—upward relocation statements

```
[1]       <RELOCATE>
[2]         <FROM>
[3]           <SOURCE><CLASS>T2</CLASS></SOURCE>
[4]           <SOURCE><CLASS>T3</CLASS></SOURCE>
[5]         </FROM>
[6]         <TO>
[7]           <DESTINATION><CLASS>T1</CLASS></DESTINATION>
[8]           <DESTINATION><CLASS>T2</CLASS></DESTINATION>
[9]         </TO>
[10]        <WHEN>
[11]          <IOTEMP Type="nrwbytes "><MIN Flags="gt">10</MIN><PERIOD>5</PERIOD></IOTEMP>
[12]        </WHEN>
[13]      </RELOCATE>

[14]      <RELOCATE>
[15]        <FROM>
[16]          <SOURCE><CLASS>T3</CLASS></SOURCE>
[17]        </FROM>
[18]        <TO>
[19]          <DESTINATION><CLASS>T2</CLASS></DESTINATION>
[20]        </TO>
[21]        <WHEN>
[22]          <IOTEMP Type="nrwbytes "><MIN Flags="gt">5</MIN><PERIOD>5</PERIOD></IOTEMP>
[23]        </WHEN>
[24]      </RELOCATE>
[25]    </RULE>

[26]  </PLACEMENT_POLICY>
```

Summary: common properties of built-in file placement policies

This appendix illustrates an instance of the access age-based with preferred files policy type created by the VEA wizard for a file system in which volumes are classified in three placement classes for ordinary (non-preferred) files (`T1`, `T2`, and `T3`), and a fourth class (`Tp`) is reserved for preferred files. Provision is made to exempt designated sets of non-preferred files from relocation if appropriate, and place them permanently on the top storage tier ("key" files) or the lowest tier ("low" files). Exemptions from file relocation are optional. To bypass exemptions, an administrator would make no entries in the wizard pages shown in Figure 6-8.

The other three built-in policies are identical in form to the access age-based with preferred files one, differing only in the relocation qualifiers, and the absence of rules for preferred files (Fragment A-1, Fragment A-2, and Fragment A-3).

Built-in policies can be created for between two and five storage tiers. The wizard adjusts the number of `RELOCATE` statements and the number of relocation source and destination placement classes in the statements appropriately.

Appendix B

A file placement policy example

This appendix includes the following topics:

- Common placement policy elements

Common placement policy elements

This appendix presents a sample file placement policy that includes common combinations of statements, clauses, selection criteria, elements, and sub-elements. The document, which in actual practice would be a single XML file, is presented in six sections for readability and to allow emphasis on individual rules and statements.

Files pinned to a storage tier

The first section, shown in Fragment B-1, contains the XML front matter, and illustrates a rule that places files of a certain naming pattern that reside in a certain directory on `tier1` volumes, and does not relocate them.

Fragment B-1 Rule for movie files

```
<?xml version="1.0"?>
<!DOCTYPE PLACEMENT_POLICY SYSTEM "/opt/VRTSfspro/config/placement_policy.dtd">
<PLACEMENT_POLICY Name="Placement_policy" Version="5.0">

  <RULE Name="Movie_Rule" Flags="none">
    <SELECT Flags="Data">
      <DIRECTORY Flags="recursive">movies</DIRECTORY>
      <PATTERN>*.wmv</PATTERN>
    </SELECT>
    <CREATE><ON><DESTINATION><CLASS>tier1</CLASS></DESTINATION></ON></CREATE>
  </RULE>
```

The rule in lines [4]-[10] of Fragment B-1 places files named according to the pattern *.wmv that reside in directory movies of the file system (relative to its mount point) or any of its sub-directories in placement class tier1 when they are created. Because the rule does not contain a RELOCATE statement, selected files are never relocated, even if they might have been selected by a later rule in the policy. Because the <ON> clause does not contain a Flags=Any modifier, creation of wmv files in these directories fails if there is no available space for them on tier1 volumes.

Multiple selection criteria

The second policy rule example illustrates both multiple SELECT statements and multiple selection criteria within a statement. Fragment B-2 contains the rule's SELECT statements.

Fragment B-2 SELECT statements for Multi-Option_Rule

```
[1]     <RULE Name="Multi-Option_Rule" Flags="none">
[2]       <SELECT Flags="Data">
[3]         <DIRECTORY Flags="recursive">ora/db</DIRECTORY>
[4]         <DIRECTORY Flags="nonrecursive">crash/dump/monthly</DIRECTORY>
[5]         <USER>user1</USER>
[6]         <USER>user2</USER>
[7]       </SELECT>
[8]       <SELECT Flags="Data">
[9]         <DIRECTORY>home/user1</DIRECTORY>
[10]        <PATTERN>*.jpg</PATTERN>
[11]        <PATTERN>my_data.txt</PATTERN>
[12]      </SELECT>
```

The first SELECT statement in lines [2]-[7] of Fragment B-2 selects files that reside either in the file system's ora/db directory, in any of the subdirectories of ora/db, or in the /crash/dump/monthly directory (but not its sub-directories), as well as belonging to either user1 or user2. Files must reside in one of the specified directories and belong to one of the specified users to be selected by this rule.

The second SELECT statement in lines [8]-[12] selects files in the home/user1 directory whose names follow the pattern *.jpg, as well as the file home/user1/my_data.txt. Files selected by this statement need not be owned by any particular user.

Multiple sources and destinations

Fragment B-3 shows the action statements for the Multi-Option_Rule whose SELECT statements are illustrated in Fragment B-2.

Fragment B-3 RELOCATE statements for Multi-Option_Rule

```
[1]       <CREATE><ON>
[2]         <DESTINATION>
[3]           <CLASS>tier1</CLASS>
[4]           <BALANCE_SIZE Units="MB">2</BALANCE_SIZE>
[5]         </DESTINATION>
[6]         <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
[7]       </ON></CREATE>
[8]       <RELOCATE>
[9]         <FROM>
[10]          <SOURCE><CLASS>tier1</CLASS></SOURCE>
[11]          <SOURCE><CLASS>tier2</CLASS></SOURCE>
[12]        </FROM>
[13]        <TO>
[14]          <DESTINATION><CLASS>tier3</CLASS></DESTINATION>
[15]          <DESTINATION><CLASS>tier4</CLASS></DESTINATION>
[16]        </TO>
[17]        <WHEN>
[18]          <SIZE Units="MB"><MIN Flags="gt">1</MIN><MAX Flags="lt">1000</MAX></SIZE>
[19]          <ACCAGE Units="days"><MIN Flags="gt">30</MIN></ACCAGE>
[20]        </WHEN>
[21]      </RELOCATE>
[22]      <RELOCATE>
[23]        <FROM><SOURCE><CLASS>tier2</CLASS></SOURCE></FROM>
[24]        <TO><DESTINATION><CLASS>tier4</CLASS></DESTINATION></TO>
[25]        <WHEN>
[26]          <IOTEMP Type="nrwbytes">
[27]            <MIN Flags="gt">2.5</MIN><PERIOD>5</PERIOD>
[28]          </IOTEMP>
[29]        </WHEN>
[30]      </RELOCATE>
[31]    </RULE>
```

The `CREATE` statement, in lines `[1]-[7]` of Fragment B-3, allocates space for selected files on `tier1` volumes if space is available, and on `tier2` volumes if not. Files allocated on `tier1` volumes are distributed across them in extents of two megabytes, but if lack of available space on `tier1` volumes forces files to be allocated on `tier2` volumes, extents are not distributed.

The first `RELOCATE` statement, in lines `[8]-[21]` of Fragment B-3, relocates selected files larger than one megabyte but smaller than 1,000 megabytes that have not been accessed for 30 days and that reside on `tier1` or `tier2` volumes to `tier3` volumes if space permits, and to `tier4` volumes if not. If neither `tier3` nor `tier4` volumes have adequate space, for a selected file that qualifies for relocation, it is not relocated, but no error is generated.

The second `RELOCATE` statement, in lines `[22]-[31]` of Fragment B-3, relocates selected files that reside on `tier2` volumes to `tier4` volumes if their five-day average I/O temperatures are greater than 2.5. Both reads and writes are included in I/O temperature computations. A file selected by this rule qualifies for relocation if the average daily amount of data read from and written to it is 2.5 times the file's size over the five-day period that immediately precedes policy enforcement.

The "Flags=Any" attribute

Fragment B-4 illustrates a rule that allocates new files if space is available anywhere in the file system's volume set.

Fragment B-4 Rule for database files

```
<RULE Name="Database_Rule" Flags="none">
  <SELECT Flags="Data"><PATTERN>*.db</PATTERN></SELECT>
  <CREATE><ON Flags=Any>
    <DESTINATION><CLASS>tier1</CLASS></DESTINATION>
  </ON></CREATE>
</RULE>
```

Files are selected by the rule in Fragment B-4 if their names follow the pattern `*.db`, regardless of the directories in which they reside or their owners. They are allocated on `tier1` volumes if space permits, but if all `tier1` volumes are fully occupied, VxFS allocates them elsewhere in the file system's volume set. Allocation and extension of files selected by this rule does not fail unless no space is available anywhere in the file system's volume set. Once allocated, files selected by this rule are not relocated. Once allocated, however, a file selected by this rule may be extended on a different volume if the volume it occupies has no free space for extension.

File deletion

Fragment B-5 illustrates a rule that automatically deletes selected files that meet certain qualifications.

Fragment B-5 Rule for selective deletion of mpg files

```
<RULE Name="MPG_Rule" Flags="none">
  <SELECT Flags="Data"><PATTERN>*.mpg</PATTERN></SELECT>
    <CREATE><ON Flags=Any>
      <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
    </ON></CREATE>
  <DELETE><FROM>
    <SOURCE><CLASS>tier1</CLASS></SOURCE>
    <SOURCE><CLASS>tier3</CLASS></SOURCE>
    <SOURCE><CLASS>tier4</CLASS></SOURCE>
  </FROM></DELETE>
  <DELETE>
    <FROM><SOURCE><CLASS>tier2</CLASS></SOURCE></FROM>
    <WHEN><ACCAGE Units="days"><MIN Flags="gt">30</MIN></ACCAGE></WHEN>
```

Fragment B-5 Rule for selective deletion of mpg files *(continued)*

```
    </DELETE>
  </RULE>
```

The rule in Fragment B-5 creates files whose names follow the pattern `*.mpg` on `tier2` volumes if space is available, and elsewhere in the file system's volume set if not. Files selected by the rule that reside on `tier1`, `tier3`, and `tier4` volumes (because no `tier1` volume had sufficient free space when they were allocated) are deleted unconditionally each time the policy is enforced. Selected files that reside on `tier2` volumes are deleted if they are inactive for 30 days.

The "catchall" rule

Fragment B-6 contains the "catchall" rule that governs placement of all files that are not selected by any of the four preceding rules. These files are created on `tier3` volumes if free space permits, and on `tier4` volumes if not. If they are accessed during the 24-hour period immediately preceding a placement policy enforcement, they are relocated to `tier1` volumes if space permits, and to `tier2` volumes if not. If no `tier1` or `tier2` space is available, files are not relocated, but no error message is generated. Once a file selected by this rule is relocated to a `tier1` or `tier2` volume, it does not relocate further because no statement in this rule applies to selected files that reside on these volumes.

Fragment B-6 "Catchall" rule

```
  <RULE Name="Catchall_Rule" Flags="data">
    <SELECT><PATTERN>*</PATTERN></SELECT>
    <CREATE><ON>
      <DESTINATION><CLASS>tier3</CLASS></DESTINATION>
      <DESTINATION><CLASS>tier4</CLASS></DESTINATION>
    </ON></CREATE>
    <RELOCATE>
      <TO>
        <DESTINATION><CLASS>tier1</CLASS></DESTINATION>
        <DESTINATION><CLASS>tier2</CLASS></DESTINATION>
      </TO>
      <WHEN><ACCAGE Units="days"><MAX Flags="lt">1</MAX></ACCAGE></WHEN>
    </RELOCATE>
  </RULE>
</PLACEMENT_POLICY>
```

Appendix C

XML document type definition (DTD) for the VxFS file placement policy document type

This appendix includes the following topics:

The Storage Foundation Placement Policy DTD

This appendix contains the XML document type definition (DTD) for file placement policy documents. Certain elements and attributes in this version of the DTD are marked as not being implemented. These represent functionality that may be provided in a future (post Version 5.0) version of the Dynamic Storage Tiering facility.

For readability, the continuous DTD XML document is presented in ten separate parts.

Fragment C-1 DTD PLACEMENT_POLICY element definition

```
<?xml version="1.0"?>
<!-- The placement policy document definition file -->
<!-- Specification for PLACEMENT_POLICY element.
     It can contain the following
           1. 0 or 1 COMMENT element
           2. 1 or more RULE elements
  -->
<!ELEMENT PLACEMENT_POLICY (COMMENT?, RULE+)>
```

Fragment C-1 DTD PLACEMENT_POLICY element definition *(continued)*

```
<!-- The attributes of PLACEMENT_POLICY element -->
        <!-- The possible and accepted values for Guid are
                1. {a5191230-1dd1-11b2-a73e-0003ba44a638} - uncanned
                2. {9d6eafe2-1dd1-11b2-a73c-0003ba44a638} - age based
                3. {992a5fc0-1dd1-11b2-a73d-0003ba44a638} - update age based
                4. {dfaa5748-1dd1-11b2-a73d-0003ba44a638} - preferred files
                5. {c094e7e2-1dd1-11b2-a73d-0003ba44a638} - activity based
             The possible and accepted values for Generation are
                1. 0 for CLI invoked at shell level
                2. 1 for policy created by local GUI
                3. > 2 for policies assigned from the central GUI
             TODO - The Guid and Generation attributes should not be documented
                        for public use.
        >

<!ATTLIST PLACEMENT_POLICY
           Name CDATA #REQUIRED
           Version "5.0" #REQUIRED
           Guid CDATA #REQUIRED
           Generation CDATA #REQUIRED

<!-- Specification for COMMENT element -->
<!ELEMENT COMMENT (#PCDATA)>
```

Fragment C-2 DTD RULE element definition

```
<!-- Specification for RULE element.
      It can contain the following
        1. 0 or 1 COMMENT element
        2. 1 or more SELECT elements
        3. 0 or 1 CREATE element
        4. 0 or more DELETE elements
        5. 0 or more RELOCATE elements

     The elements must appear in the above order, particularly,
     DELETE elements, if any, must preceed RELOCATE elements, if any.
     If any of the DELETE elements triggers an action, subsequent
     elements (DELETE and/or RELOCATE elements, if any) will not be
     processed.
  -->
<!ELEMENT RULE (COMMENT?, SELECT+, CREATE?, DELETE*, RELOCATE*)>
<!-- The attributes of RULE element -->
        <!-- The possible and accepted values for Flags are
                1. data
                2. metadata
                3. checkpoint
                4. log

             If Flags indicate "metadata", SELECT element must not
             contain any DIRECTORY, PATTERN, USER or GROUP elements
             in such SELECT element. There must not be any DELETE
             element in the enclosing RULE element either.

             If Flags indicate "checkpoint", SELECT element must not
             contain any DIRECTORY, PATTERN, USER or GROUP elements
```

Fragment C-2 DTD RULE element definition *(continued)*

```
             in such SELECT element. There must not be any RELOCATE or
             DELETE element in the enclosing RULE element either.

             If Flags indicate "log", SELECT element must not
             contain any DIRECTORY, PATTERN, USER or GROUP elements
             in such SELECT element. There must not be any RELOCATE
             element in the enclosing RULE element either.
             TODO - Remove metadata, checkpoint and log references from
                    public use
          -->
<!ATTLIST RULE
        Name CDATA #REQUIRED
        Flags ("data"|"metadata"|"checkpoint"|"log") #REQUIRED
>
```

Fragment C-3 DTD SELECT element definition

```
<!-- Specification for SELECT element. This describes selection criteria.
     It can contain the following
     1. 0 or 1 COMMENT elements
     2. 0 or more DIRECTORY elements
     3. 0 or more PATTERN elements
     4. 0 or more USER elements
     5. 0 or more GROUP elements
     The elements can appear in any order.
  -->
<!ELEMENT SELECT (COMMENT?, DIRECTORY*, PATTERN*, USER*, GROUP*)>
<!-- The attributes of SELECT element -->
<!ATTLIST SELECT
        Name CDATA
>

<!-- Specification for DIRECTORY element
        The DIRECTORY element takes a path relative to the
        mount point. So if the intention is to sweep from
        /db/finance/data and /db is the mount point,
        DIRECTORY element should contain finance/data
        Only one value can be specified per element.
  -->
<!ELEMENT DIRECTORY (#PCDATA)>
<!-- The attributes of DIRECTORY element -->
        <!-- The possible and accepted values for Flags are
                1. recursive
                2. nonrecursive

                If a given directory appears in more than one RULE,
                all such DIRECTORY elements must all be recursive or
                nonrecursive but can not be a combination. If no DIRECTORY
                element is specified, all the files under the mount point
                will be selected.
         -->
<!ATTLIST DIRECTORY
         Flags ("recursive"|"nonrecursive") #REQUIRED
>
```

Fragment C-3 DTD SELECT element definition *(continued)*

```
<!-- Specification for PATTERN element
        The PATTERN can be a full name of a file, i.e., can not contain
        "/" characters. Or it can have a '*' character. The first '*'
        character will be considered as wild character and any other
        character, including a second '*' as literals.
        Only one value can be specified per element.
  -->
<!ELEMENT PATTERN (#PCDATA)>

<!-- Specification for USER element
        The USER is a name string of the unix domain user
        Only one value can be specified per element.
 -->
<!ELEMENT USER (#PCDATA)>

<!-- Specification for GROUP element
        The GROUP is a name string of the unix domain group
        Only one value can be specified per element.
  -->
<!ELEMENT GROUP (#PCDATA)>
```

Fragment C-4 DTD CREATE element definition part 1

```
<!-- Specification for CREATE element. This describes creation criteria.
     It can contain the following
     1. 0 or 1 COMMENT element
     2. 1 ON element

     The order of elements may be significant in future
  -->
<!ELEMENT CREATE (COMMENT?, ON)>
<!ATTLIST CREATE
        Name CDATA
        Flags CDATA "none"
>

<!-- Specification for ON element. This describes location criteria.
     It can contain the following
     1. 0 or more DESTINATION elements
     Though zero DESTINATION elements is defined in grammar, current
     implementation requires at least on DESTINATION.
  -->
<!ELEMENT ON (DESTINATION*)>
<!-- The attributes of ON element -->
        <!-- The possible and accepted values for Flags is
                1. any
                If this attribute is set, there may or may not be any CLASS
                elements in the DESTINATION elements under the ON element.
                If any of the DESTINATION elements have CLASS element, such
                CLASSes in the file system would be used first before other
                placement class storage is used.
         -->
<!ATTLIST ON
```

Fragment C-4 DTD CREATE element definition part 1 *(continued)*

```
         Name CDATA
         Flags ("any")
>
```

Fragment C-5 DTD CREATE element definition part 2

```
<!-- Specification for DESTINATION element. This describes target location.
     It can contain the following
     1. 0 or 1 CLASS element
     2. 0 or 1 PERCENT element
     3. 0 or 1 BALANCE_SIZE element
  -->
<!ELEMENT DESTINATION (CLASS?, PERCENT?, BALANCE_SIZE?)>
<!-- The attributes of DESTINATION element -->
        <!-- The possible and accepted values for Flags
                (THIS IS NOT IMPLEMENTED)  [
                1. disallow
                If this 'disallow' is set, there must not be any PERCENT or
                BALANCE_SIZE elements in such DESTINATION element but there
                must be a CLASS element. There must not be any RELOCATE and
                DELETE statements in the enclosing RULE element either.
        -->
<!ATTLIST DESTINATION
        Name CDATA
        Flags ("disallow")
>

<!-- Specification for CLASS element
        The file system resides on a multi-component volume set.
        Each volume in the volume set will be in what is called a
        placement class. The placement classes are implemented as tags
        on the volumes. These tags are organized into a hierarchy prefix.
        The placement policy uses the vxfs.placement_class. prefix.
        The CLASS element specifies the placement class of the
        underlying storage, without the prefix. For example, if a
        volume has a placement class of vxfs.placment_class.gold
        then gold would be the value of CLASS element.
 -->
<!ELEMENT CLASS (#PCDATA)>

<!-- Specification for PERCENT element
         (THIS IS NOT IMPLEMENTED)
        If the PERCENT element is in DESTINATION element, it determines
        how much of its CLASS can be filled up with the files selected
        by a given RULE.

        If the PERCENT element is in SOURCE element, it determines
        how much of its CLASS can be emptied when the files are relocated
        our deleted from it.
 -->
<!ELEMENT PERCENT (#PCDATA)>

<!-- Specification for BALANCE_SIZE element
        Multiple volumes may have the same placement class and there can
```

Fragment C-5 DTD CREATE element definition part 2 *(continued)*

```
        be multiple DESTINATIONs (hence CLASSes) in a given ON (and TO)
        element. If a BALANCE_SIZE is specified for a given CLASS,
        the usage of volumes of that given placement class will be used
        evenly by allocating BALANCE_SIZE amount of space for each
        volume for each allocation.
 -->
<!ELEMENT BALANCE_SIZE (#PCDATA)>
<!-- The attributes of BALANCE_SIZE element -->
        <!-- The possible and accepted values for Units are
                1. bytes
                2. KB
                3. MB
                4. GB
         -->
<!ATTLIST BALANCE_SIZE
        Units ("bytes"|"KB"|"MB"|"GB") #REQUIRED
>
```

Fragment C-6 DTD DELETE element definition

```
<!-- Specification for DELETE element. This describes deletion criteria.
     It can contain the following
     1. 0 or 1 COMMENT element
     2. 0 or 1 FROM element
     3. 0 or 1 WHEN element
  -->
<!ELEMENT DELETE (COMMENT?, FROM?, WHEN?)>
<!ATTLIST DELETE
 Name CDATA
 Flags CDATA "none"
>
```

Fragment C-7 DTD RELOCATE element definition part 1

```
<!-- Specification for RELOCATE element. This describes relocation criteria.
     It can contain the following
     1. 0 or 1 COMMENT element
     2. 0 or 1 FROM element
     3. 1 TO element
     4. 0 or 1 WHEN element

     The order of TO elements is significant. Earlier CLASSes would be
     used before the latter ones.
  -->
<!ELEMENT RELOCATE (COMMENT?, FROM?, TO, WHEN?)>
<!ATTLIST RELOCATE
        Name CDATA
        Flags CDATA "none"
>

<!-- Specification for FROM element. This describes source criteria.
     It can contain the following
```

Fragment C-7 DTD RELOCATE element definition part 1 *(continued)*

```
     1. 1 or more SOURCE elements
  -->
<!ELEMENT FROM (SOURCE+)>
<!ATTLIST FROM
        Name CDATA
        Flags CDATA "none"
>

<!-- Specification for SOURCE element. This describes source location.
     It can contain the following
     1. 1 CLASS element
     2. 0 or 1 PERCENT element
  -->
<!ELEMENT SOURCE (CLASS, PERCENT?)
<!ATTLIST SOURCE
        Name CDATA
        Flags CDATA "none"
>

<!-- Specification for TO element. This describes destination criteria.
     It can contain the following
     1. 1 or more DESTINATION elements
  -->
<!ELEMENT TO (DESTINATION+)>
<!ATTLIST TO
 Name CDATA
 Flags CDATA "none"
>
```

Fragment C-8 DTD RELOCATE element definition part 2

```
<!-- Specification for WHEN element. This describes relocation specifiers.
     It can contain the following
     1.  0 or 1 SIZE element
     2.  0 or 1 ACCAGE element
     3.  0 or 1 MODAGE element
     4.  0 or 1 IOTEMP element
     5.  0 or 1 ACCESSTEMP element

     The order of elements is significant.
  -->
<!ELEMENT WHEN (SIZE?, ACCAGE?, MODAGE?, IOTEMP?, ACCESSTEMP?>
<!ATTLIST WHEN
        Name CDATA
        Flags CDATA "none"
>

<!-- Specification for SIZE element -->
     It can contain the following
     1.  0 or 1 MIN element
     2.  0 or 1 MAX element
  -->
<!ELEMENT SIZE (MIN?, MAX?)>
```

Fragment C-8 DTD RELOCATE element definition part 2 *(continued)*

```
<!-- The attributes of SIZE element -->
        <!-- The possible and accepted values for Prefer are
                (THIS IS NOT IMPLEMENTED)
                1. low
                2. high
             The possible and accepted values for Units are
                1. bytes
                2. KB
                3. MB
                4. GB
         -->
<!ATTLIST SIZE
 Prefer ("low"|"high")
 Units ("bytes"|"KB"|"MB"|"GB") #REQUIRED
>

<!-- Specification for ACCAGE element -->
     It can contain the following
     1.  0 or 1 MIN element
     2.  0 or 1 MAX element
  -->
<!ELEMENT ACCAGE (MIN?, MAX?)>
<!-- The attributes of ACCAGE element -->
        <!-- The possible and accepted values for Prefer are
         (THIS IS NOT IMPLEMENTED)
                1. low
                2. high
             The possible and accepted values for Units are
                1. hours
                2. days
        -->
<!ATTLIST ACCAGE
        Prefer ("low"|"high")
        Units (""hours"|"days") #REQUIRED
>
```

Fragment C-9 DTD RELOCATE element definition part 3

```
!-- Specification for MODAGE element -->
     It can contain the following
     1.  0 or 1 MIN element
     2.  0 or 1 MAX element
  -->
<!ELEMENT MODAGE (MIN?, MAX?)>
<!-- The attributes of MODAGE element -->
        <!-- The possible and accepted values for Prefer are
                (THIS IS NOT IMPLEMENTED)
                1. low
                2. high
             The possible and accepted values for Units are
                1. hours
                2. days
         -->
<!ATTLIST MODAGE
```

Fragment C-9 DTD RELOCATE element definition part 3 *(continued)*

```
        Prefer ("low"|"high")
        Units (""HOURS"|"days") #REQUIRED
>
<!-- Specification for IOTEMP element -->
     The value of IOTEMP represents bytes read (nrbytes),
     bytes written (nwbytes) or bytes transferred, i.e.,
     read and written (nrwbytes), divided by the size of the
     file, over a specified PERIOD (in days).
     It can contain the following
     1. 0 or 1 MIN element
     2. 0 or 1 MAX element
     3. 1 PERIOD element
  -->
<!ELEMENT IOTEMP (MIN?, MAX?, PERIOD)>
<!-- The attributes of IOTEMP element -->
        <!-- The possible and accepted values for Prefer are
                (THIS IS NOT IMPLEMENTED)
                1. low
                2. high
         -->
        <!-- The possible and accepted values for Type are
                1. nrbytes
                2. nwbytes
                3. nrwbytes
         -->
<!ATTLIST IOTEMP
        Prefer ("low"|"high")
        Type ("nrbytes"|"nwbytes"| "nrwbytes") #REQUIRED
>
<!-- Specification for ACCESSTEMP element -->
     The value of ACCESSTEMP represents times read (nrbytes),
     times written (nwbytes) or times access i.e.,
     read and written (nrws) over a specified PERIOD (in days).
     It can contain the following
     1. 0 or 1 MIN element
     2. 0 or 1 MAX element
     3. 1 PERIOD element
  -->
<!ELEMENT ACCESSTEMP (MIN?, MAX?, PERIOD)>
<!-- The attributes of ACCESSTEMP element -->
        <!-- The possible and accepted values for Prefer are
                (THIS IS NOT IMPLEMENTED)
                1. low
                2. high
         -->
        <!-- The possible and accepted values for Type are
                1. nreads
                2. nwrites
                3. nrws
         -->
<!ATTLIST ACCESSTEMP
        Prefer ("low"|"high")
        Type ("nreads"|"nwrites"| "nrws") #REQUIRED
>
```

Fragment C-10 DTD RELOCATE element definition part 4

```
<!-- Specification for MIN element -->
<!ELEMENT MIN (#PCDATA)>
<!-- The attributes of MIN element -->
          <!-- The possible and accepted values for Flags are
          1. gt for >
          2. eq for =
          3. gteq for >=
          -->
<!ATTLIST MIN
          Flags ("gt"|"eq"|"gteq") #REQUIRED
>

<!-- Specification for MAX element -->
<!ELEMENT MAX (#PCDATA)>
<!-- The attributes of MAX element -->
          <!-- The possible and accepted values for Flags are
          1. lt for <
          2. lteq for <=
          -->
<!ATTLIST MAX
          Flags ("lt"|"lteq") #REQUIRED
>

<!-- Specification for PERIOD element -->
<!ELEMENT PERIOD (#PCDATA)>
<!ATTLIST PERIOD
          Units ("days") #REQUIRED
>
```

Using Dynamic Storage Tiering

The Veritas Storage Foundation Dynamic Storage Tiering facility takes advantage of multi-tier storage by constantly aligning files' storage physical locations with policies that express changing business needs, without recurring administrative overhead or "time to first byte" penalties when files are accessed. The case for Dynamic Storage Tiering is simple:

Opportunity: Multiple online storage tiers create an opportunity for enterprises to reduce their overall cost of storage by relegating less critical or less active data to lower cost storage devices.

Feasibility: The hardware technology and virtualization options available today make it possible to tailor online storage to enterprise data requirements, whether they are expressed in terms of cost, availability, and performance, or in terms of other business criteria such as organizational responsibility or profitability.

Paradigm: Files are an obvious unit for managing relationships between data and storage tiers. Effective utilization of multi-tier storage means that files are created on the most appropriate type of storage, and relocated as their properties change during their lifecycles.

Limitations of conventional mechanisms: The two conventional mechanisms for utilizing multi-tier storage are ad hoc relocation and hierarchical storage management (HSM). Ad hoc relocation is labor-intensive and fragile. HSM is limited to inactivity-based policies and incurs run-time delays when applications access migrated files.

This book describes how Dynamic Storage Tiering works, shows how administrators can use it to automatically optimize online storage utilization, and presents some common use cases. It is a must for architects and administrators concerned with controlling storage cost without sacrificing the quality of service delivered to applications and users.

symantec™

www.symantec.com

52000

9 780321 446107

ISBN 0-321-44610-0

$20.00 US $27.00 CANADA