Veritas Storage Foundation™ Intelligent Storage Provisioning Administrator's Guide

HP-UX 11i v3

5.0

Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide

Veritas Storage Foundation 5.0

Symantec Corporation
20330 Stevens Creek Blvd.
Cupertino, CA 95014
www.symantec.com

Third-party legal notices

Third-party software may be recommended, distributed, embedded, or bundled with this Symantec product. Such third-party software is licensed separately by its copyright holder. All third-party copyrights associated with this product are listed in the accompanying release notes.

HP-UX is a registered trademark of Hewlett-Packard Development Company, L.P.

Licensing and registration

Veritas Storage Foundation is a licensed product. See the *Veritas Storage Foundation Installation Guide* for license installation instructions.

Technical support

For technical assistance, visit
http://www.symantec.com/enterprise/support/assistance_care.jsp

and select phone or email support. Use the Knowledge Base search feature to access resources such as TechNotes, product alerts, software downloads, hardware compatibility lists, and our customer email notification service.

Contents

Chapter 1 Understanding ISP

Chapter 1

Understanding ISP

Intelligent Storage Provisioning (ISP) allows you to organize and manage your physical storage by creating virtual storage devices, *application volumes*. You can use such volumes in the same way as traditional volumes in Veritas™ Volume Manager (VxVM) by Symantec.

ISP creates application volumes from available storage with the capabilities you specify. It selects storage by consulting the externally-defined rule base for creating volumes, and compares this with the properties of the storage that is available.

ISP provides the following functionality:

- Creation and removal of application volumes.
- Organizing storage by grouping into storage pools.
- Resizing a volume while it is online.
- Moving or evacuating subdisks of a volume.
- Adding mirrors and columns to a volume while it is online.
- Removing mirrors and columns from a volume while it is online.
- Relocating a subdisk of a volume while it is online.
- Changing the capabilities of a volume while it is online.
- Creating volume snapshots using software or hardware.
- Reallocation of storage to preserve the capabilities of a volume.

ISP is capable of understanding Storage Area Network (SAN) topology and of efficiently using the available intelligent storage. ISP interacts with other Veritas components, where these are available, such as the Veritas Array Integration Layer (VAIL) and the SAN Access Layer (SAL), to take appropriate actions when configuring intelligent storage in a SAN environment.
See “Configuring ISP to work with SAL” on page 215.

Note: Products and features such as dynamic multipathing (DMP), RAID-5, SAL, snapshots, and VAIL require licenses in addition to the base license.

This book describes the command-line interface to ISP, and the language that is used for writing new rules, capabilities and templates. For a description of the graphical interface to ISP that is provided by the VERITAS Enterprise Administrator (VEA), see the *VERITAS Enterprise Administrator User's Guide* and VEA online help.

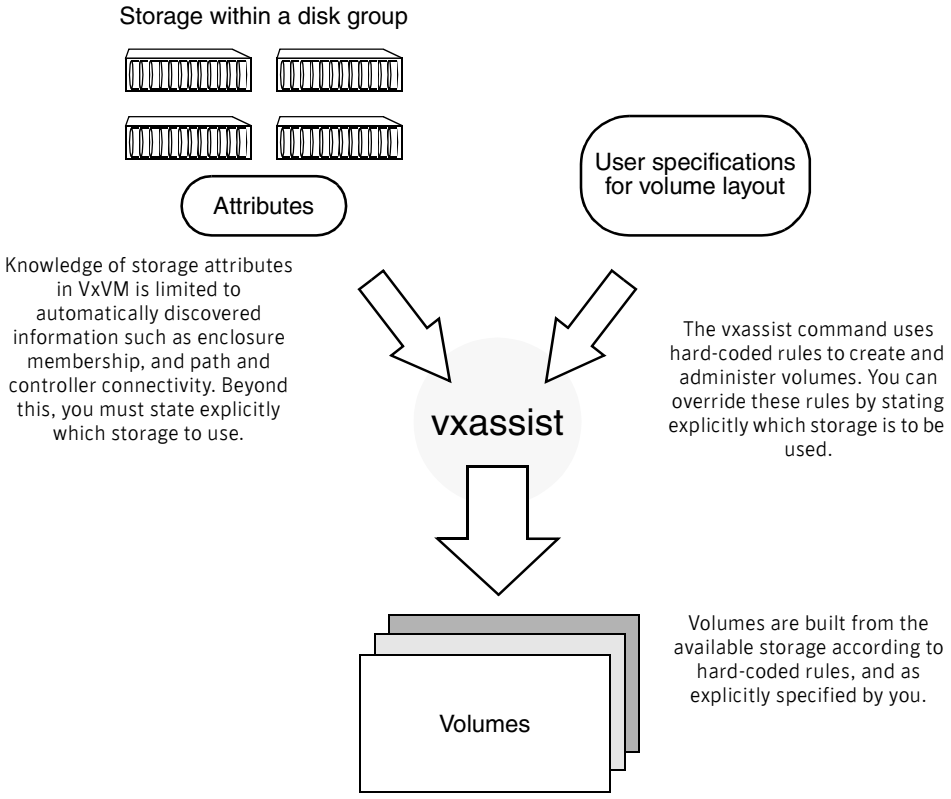
For more information about how you can apply ISP in different scenarios such as implementing storage tiers, offhost processing, or remote mirroring between different locations, see the *Veritas Storage Foundation Intelligent Storage Provisioning Solutions Guide*.

The benefits of ISP

When creating a volume in Veritas Volume Manager in previous releases, you could specify the disk storage on which to lay out its various parts, subdisks, plexes, and so on. In specifying the storage to be used, you had to take into account the tolerance of a volume to failure of any component of the storage infrastructure, and how the specified layout affected I/O performance and reliability of service. For small installations with a few tens of disks in relatively low-specification arrays, you could either specify the storage layout manually to commands such as `vxassist`, or rely on `vxassist` to choose appropriate storage based on general layout specification, such as "mirror across controllers" and "mirror across enclosures," and using the set of heuristic rules that were hard-coded within `vxassist`.

The traditional model for allocating storage to volumes is shown in Figure 1-1. This illustrates that, although some storage attributes are known to VxVM, you must do most of the work in deciding how to lay out the storage if you are to create a volume with the desired performance, reliability and fault tolerance.

Figure 1-1 Traditional model for creating and administering volumes in Veritas Volume Manager



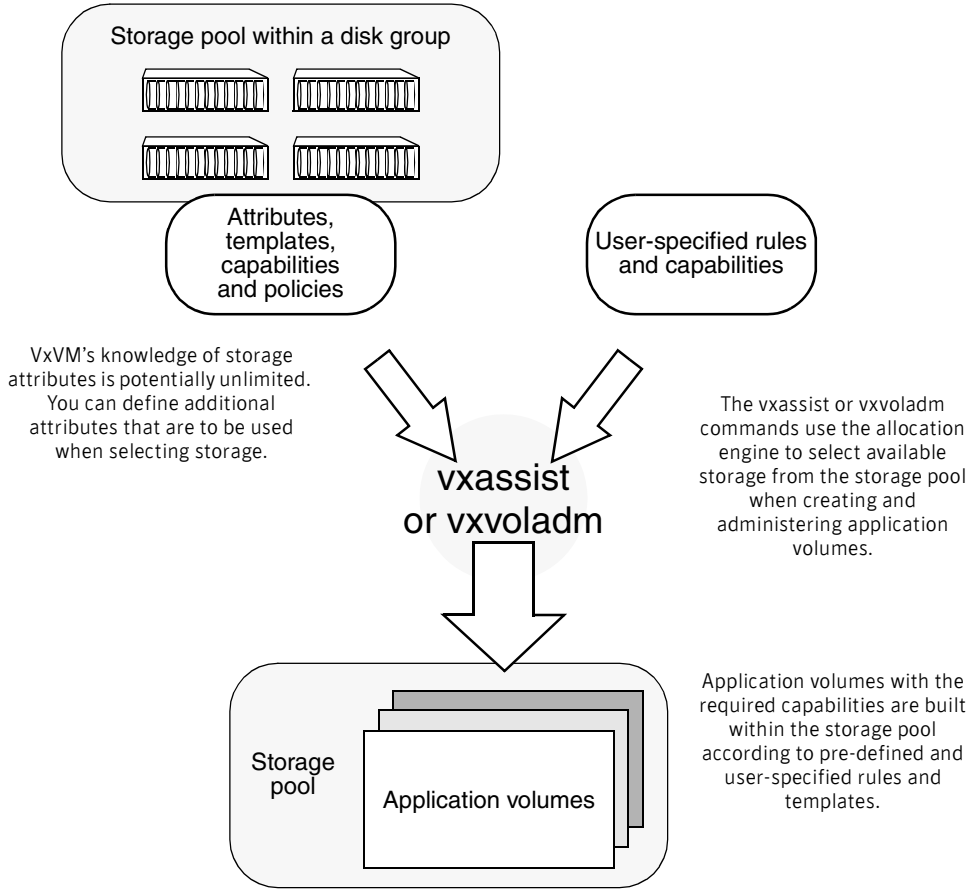
When intelligent disk arrays are used, many sophisticated features, such as RAID capabilities, snapshot facilities, and remote replication, are provided by logical unit storage devices, LUNs, that are exported by the disk array. Such devices may or may not have ways of making their attributes known to VxVM. In any case, you may be presented with hundreds or thousands of LUNs connected over a SAN.

Allocating storage to volumes when faced with a potentially large number of devices with widely varying and possibly hidden properties is a daunting task to perform manually. ISP aids you in managing large sets of storage by providing an allocation engine that chooses which storage to use based on the capabilities that you specify for the volumes to be created.

Figure 1-2 illustrates how ISP improves on the traditional model for creating volumes. The main differences are that the set of information about the

available storage is potentially unlimited, and the set of rules that the allocation engine uses to choose storage is defined externally to commands such as `vxassist` and `vxvoladm`.

Figure 1-2 How ISP enhances volume management



Support for ISP in the vxassist command

In release 5.0 of VxVM, it is now possible to use the `vxassist` command to create and administer ISP volumes. The `vxassist` command now accepts the same specification of templates, capabilities and rules as the `vxvoladm` command, and a set of `vxassist` storage specification attributes are automatically translated into equivalent ISP rules.

If the `-o intent` option is specified, `vxassist` creates an ISP volume, and it also sets up a storage pool in the disk group if one does not already exist. If a storage pool already exists in a disk group, the `vxassist` command attempts to create an ISP volume unless you specify the `-o nointent` option.

All operations in this book that use the `vxassist` command have an equivalent `vxvoladm` command, which is obtained by substituting `vxvoladm` for `vxassist`. Any other arguments to the command remain the same. A `vxvoladm` command is shown if there is no equivalent `vxassist` command.

Summary of the benefits of using ISP

The following list summarizes the main benefits that Veritas ISP provides over the existing storage allocation features in `vxassist`:

- Storage is automatically allocated based on abstract requirements such as the desired capabilities of a volume.
- Prefabricated capabilities that are provided by vendor-specific features of intelligent storage arrays can be encoded as storage attributes, and used to allocate storage.
- Volumes can be created or grown in batch mode safe in the knowledge that ISP will balance the requirements of all volumes.
- All ISP operations preserve the original intent of the volumes. There is no possibility that operations such as grow, evacuate, mirror, or add column can accidentally degrade the reliability or performance capabilities of a volume.
- ISP is SAN-aware and understands SAN attributes. It is also capable of using VAIL to learn the capabilities of LUNs.
- Disk tags, administered using the `vxdisk` command or the VEA graphical user interface, allow you to define attributes for LUNs that lie outside their discovered hardware characteristics, and to assign values to these attributes.

Limitations of ISP

The following features of `vxassist` are not currently supported for ISP volumes:

- The `vxassist` utility includes a number of hard-coded rules that it uses when selecting storage. For example, `vxassist` may configure objects on separate controllers without being instructed to do so. ISP requires that the selection of storage is made explicit through rules, capabilities and templates.
- Disk group split and join is supported at the level of storage pools. A snapshot of an application volume should be created within a clone pool if it is to be moved between disk groups.

Frequently asked questions about ISP

- *What is the meaning of the new concepts that are introduced by ISP?*
  ISP introduces several new concepts in addition to those that are used with traditional Veritas Volume Manager. New concepts, including storage pools, capabilities, rules, volume templates and volume intent, are described in "About ISP concepts" on page 33.
- *What is the relationship between a storage pool and a volume template?*
  Storage pools contain disks, VxVM objects such as volumes, and a set of volume templates. A storage pool is defined by the volume templates that it contains. The ISP Configuration Database contains a number of storage pool definitions that you can use to create a storage pool object in VxVM. Each definition contains a list of volume templates and the default policy settings for the pool definition. When you create a storage pool object in a disk group from a storage pool definition, these volume templates and policies also get installed.
- *When do I need to specify a template set?*
  A storage pool contains volume templates that define its characteristics. A template set is simply a collection of related volume templates that you can associate with a storage pool.
- *When do I need to specify a storage pool set?*
  A disk group that you want to use with ISP must be configured to contain one data storage pool, and optionally one or more clone storage pools. You can use a storage pool set definition to organize a disk group so that it contains data and clone storage pools with well-defined characteristics.
- *If I create a storage pool using a storage pool definition, can I later create a volume in that pool using a volume template other than those that are associated with the pool?*

The answer can be illustrated by an example. Suppose you create a storage pool using the `mirrored_volumes` pool definition. This installs volume templates that allow you to create mirrored volumes and volumes with similar characteristics in the pool. If you attempted to create a striped volume in the pool, the resulting volume is mirrored as well as striped. However, you are not constrained from creating volumes with other characteristics. You can use rules to bypass restrictions that are imposed by higher-level abstractions like volume templates and capabilities.

Bear in mind that creating a storage pool from a storage pool definition does not uniquely determine the capabilities of volumes that you create in the pool. Only when you create a volume can you specify its capabilities and ensure consistency between the volumes in a pool.

- *What does "prefabricated" mean as used in volume templates?*
  The term *prefabricated* implies that the characteristics of a volume are implemented using hardware rather than software. For example, `PrefabricatedRaid5` implies the use of RAID-5 LUNs that have been set up in an array's hardware, rather than being implemented as a VxVM RAID-5 volume.
- *Can I use both hardware and software RAID volumes in a storage pool?*
  A storage pool is usually configured either for prefabricated RAID devices or for VxVM volumes that are created in software. This provides consistency in performance and failure tolerance within the pool. If you add both hardware and software RAID templates to a pool, ISP allocates templates that are appropriate to the capabilities of the volumes that you specify.
- *How does ISP discover LUN hardware characteristics?*
  ISP relies on the Veritas Array Integration Layer (VAIL) to provide detailed information on LUN characteristics via array-specific modules.
- *How do I restrict allocation to storage from certain vendors, for example EMC BCV or Hitachi?*
  By default, ISP uses LUNs with similar characteristics for allocating storage. If insufficient storage is available, it relaxes this constraint. You can make the constraint mandatory by specifying the capability `ArrayProductId` when creating a volume. This forces ISP to allocate storage on LUNS that share the same product identifier. Alternatively, you can use the capability `ConfineToSimilarStorage`. This makes ISP use LUNs from the same vendor, but allows the product identifiers of these LUNs to differ.

- *When I create a volume, ISP uses space on the same disks unless I choose other disks from the storage pool manually. Why doesn't ISP automatically spread the volumes across the other disks in the storage pool to enhance I/O performance?*
  ISP is tuned to use as few disks as possible. In any case, I/O performance depends on many factors: the way storage is connected to the system, the inherent capabilities of the storage, how volumes are configured and how they share storage, the type of I/O requests made by applications, and the amount of I/O from/to each volume. For example, allocating two volumes to a single disk array that has a large cache would probably provide better overall I/O performance than placing one volume on the disk array and the other on a JBOD. If necessary, you can configure smaller storage pools in separate disk groups to restrict the allocation of storage. Alternatively, you can explicitly specify the storage that can or cannot be allocated to a volume.

Administration roles in ISP

The administration of ISP can seem overwhelming when compared with the traditional administration model in Veritas Volume Manager. To simplify matters, it is useful to think in terms of three levels of administration based on level of knowledge of ISP, and frequency of application of this knowledge:

- Basic administration tasks
- Advanced administration tasks
- Expert administration tasks

Basic administration tasks are performed most often and require the least knowledge of ISP to perform. Expert storage administration tasks are performed least often and require the most knowledge of ISP to perform.

Basic administration tasks

Basic administration tasks include creating storage pools, and creating and administering volumes. This includes performing tasks such as adding or removing mirrors or columns from volumes, resizing volumes, and creating and using volume snapshots.

The system administrator's role requires the following knowledge of ISP:

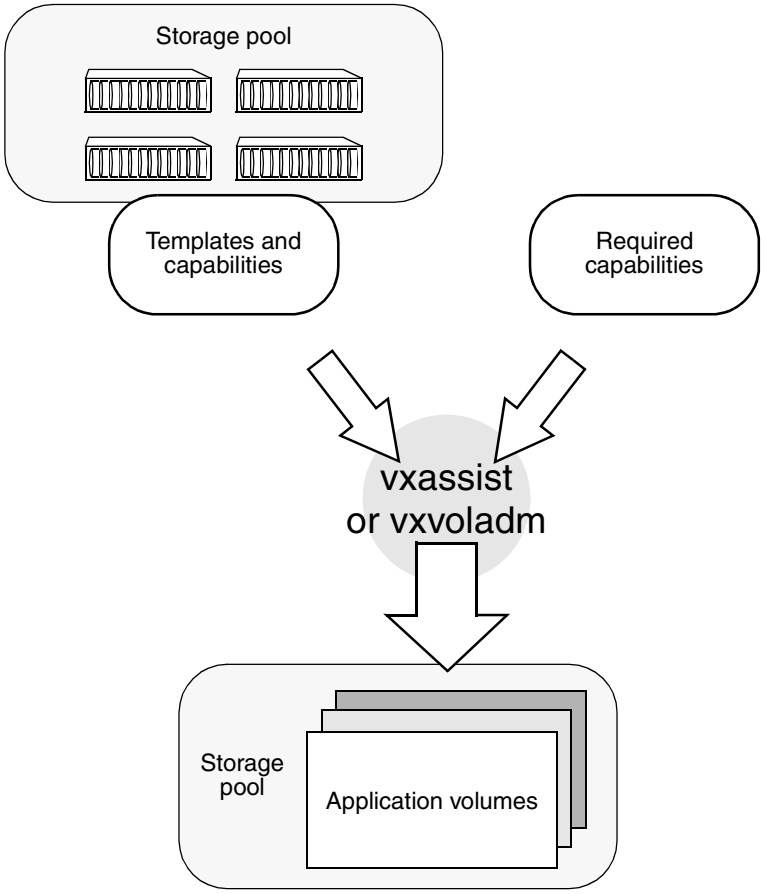
- Familiarity with the meaning of the terms disk group, storage pool, template, user template, application volume, and capabilities such as reliability, performance and fault tolerance.

- What templates and user templates are available for use in creating volumes with the required capabilities.
- What templates and user templates are associated with the disk groups in which you will create volumes.
- How to use the VEA or the `vxassist` and `vxvoladm` commands to perform your tasks.

As a system administrator, you can use either the VEA or the `vxassist` and `vxvoladm` utilities to perform your tasks. The VEA graphical interface is most suitable for day-to-day administration. The `vxassist` and `vxvoladm` utilities are primarily intended for use in administration scripts.

The use of the `vxassist` or `vxvoladm` command to create application volumes is illustrated in Figure 1-3.

Figure 1-3 Creation of application volumes



System administration tasks are described in the following chapters:

- "Creating application volumes" on page 61
- "Administering application volumes" on page 75
- "Administering instant snapshots" on page 93

Advanced administration tasks

Advanced administration tasks include administering storage pools and user templates, creating clone pools, and creating volumes with additionally specified rules.

The system administrator's role requires the following knowledge of ISP:

- Knowledge of basic administration tasks.
  See "Basic administration tasks" on page 22.
- Familiarity with the meaning of the terms LUN, attribute, policy, intent, and rules such as those for confining or excluding how storage is allocated to new volumes.
- How to create or modify user templates, and then associate these with disk groups and storage pools.
- How to use the VEA or the command-line utilities to perform your tasks.

As a storage administrator, you can use either the VEA or command-line utilities to perform your tasks.

The VEA interface is most suitable for day-to-day administration. As illustrated in Figure 1-4, the command-line utilities are primarily intended for use in administration scripts, but can also be used to compile and install user templates.

Figure 1-4 Creation of user templates

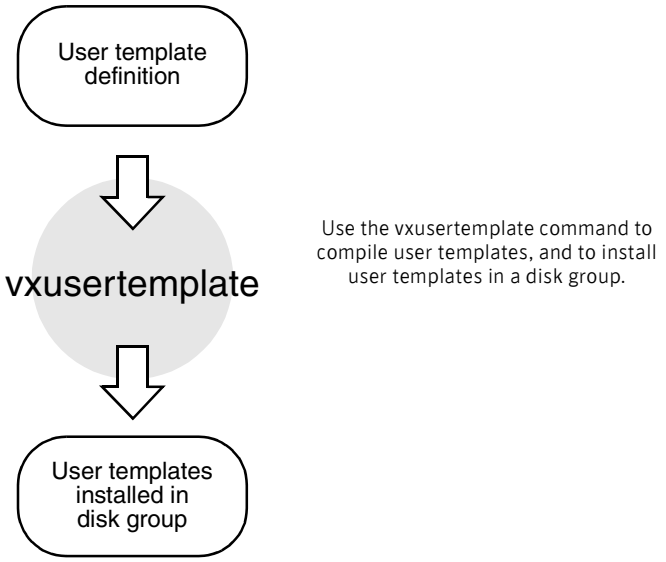
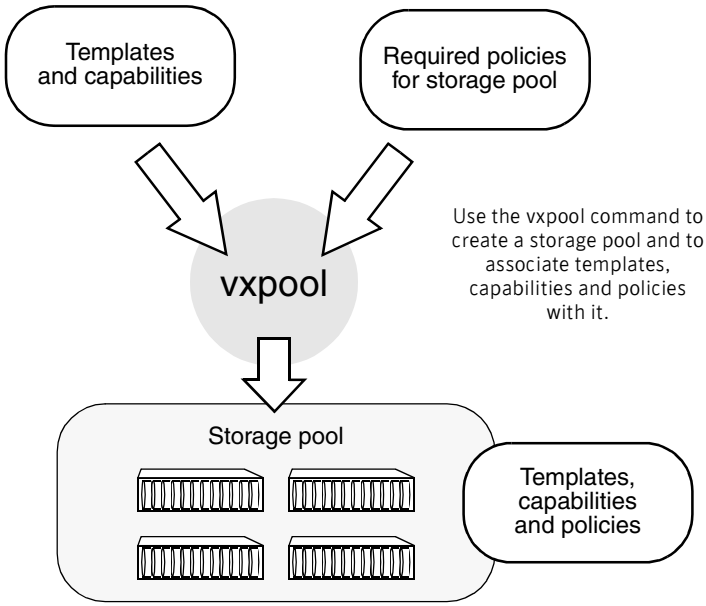


Figure 1-5 illustrates that the `vxpool` command is used to create storage pools with associated policies, templates and capabilities.

Figure 1-5 Creation of storage pools



System administration tasks are described in the following chapters:

- "Storage pools" on page 49
- "Creating and modifying user templates" on page 131

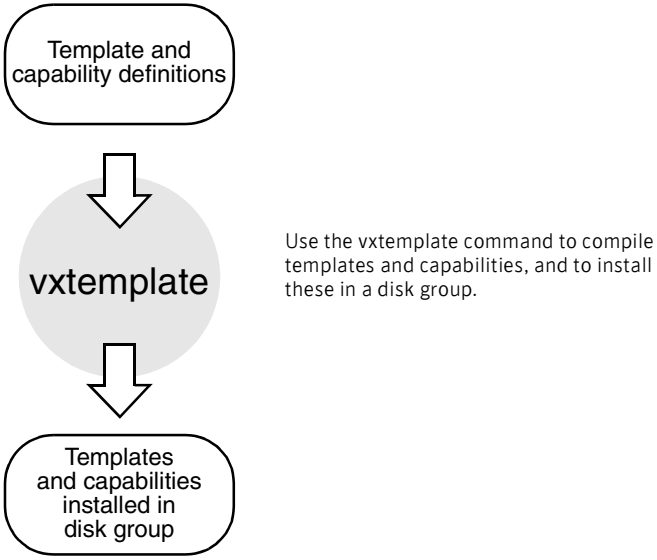
Expert administration tasks

Expert administration tasks include creating new attributes to describe storage features, and designing and creating new templates and capabilities.

The storage administrator's role requires the following knowledge of ISP:

- Knowledge of basic and advanced administration tasks.
  See "Basic administration tasks" on page 22.
  See "Advanced administration tasks" on page 25.
- Familiarity with the ISP rule specification language, and how to use it to write rules, capabilities and templates.
- How to use the VEA or the command-line utilities to perform your tasks. You can use any suitable text editor to create template definition files. The command-line utilities are primarily intended for use in administration scripts, but can also be used to compile and install templates and capabilities as illustrated in Figure 1-6.

Figure 1-6 Creating templates and capabilities



Advanced storage administration tasks are described in the following chapters:

- "Administering volume templates and other configuration elements" on page 121
- "Using capabilities, templates and rules" on page 137
- "ISP language definition" on page 159

Sample ISP deployments

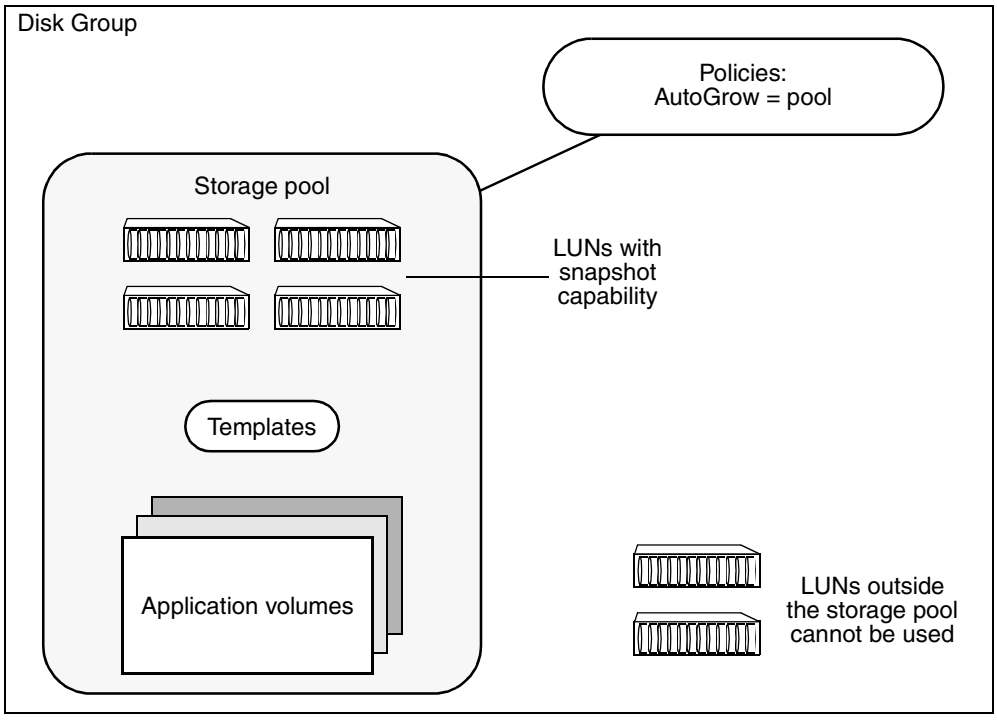
This section contains high-level examples of how you can configure volume creation using ISP.

Using storage pool policies

As described in "About storage pool policies" on page 37, a storage pool's policies affect how it allocates LUNs to create new volumes, and how it uses templates. By selecting which templates and LUNs are associated with a storage pool, and setting appropriate policies on the storage pool, you can control how storage is allocated to volumes. For example, a storage pool might be composed entirely of LUNs that have a hardware snapshot capability (for example, EMC Business Continuity Volumes).

Setting the `AutoGrow` policy to `pool` ensures that the entire storage pool remains snapshot-capable as illustrated in Figure 1-7.

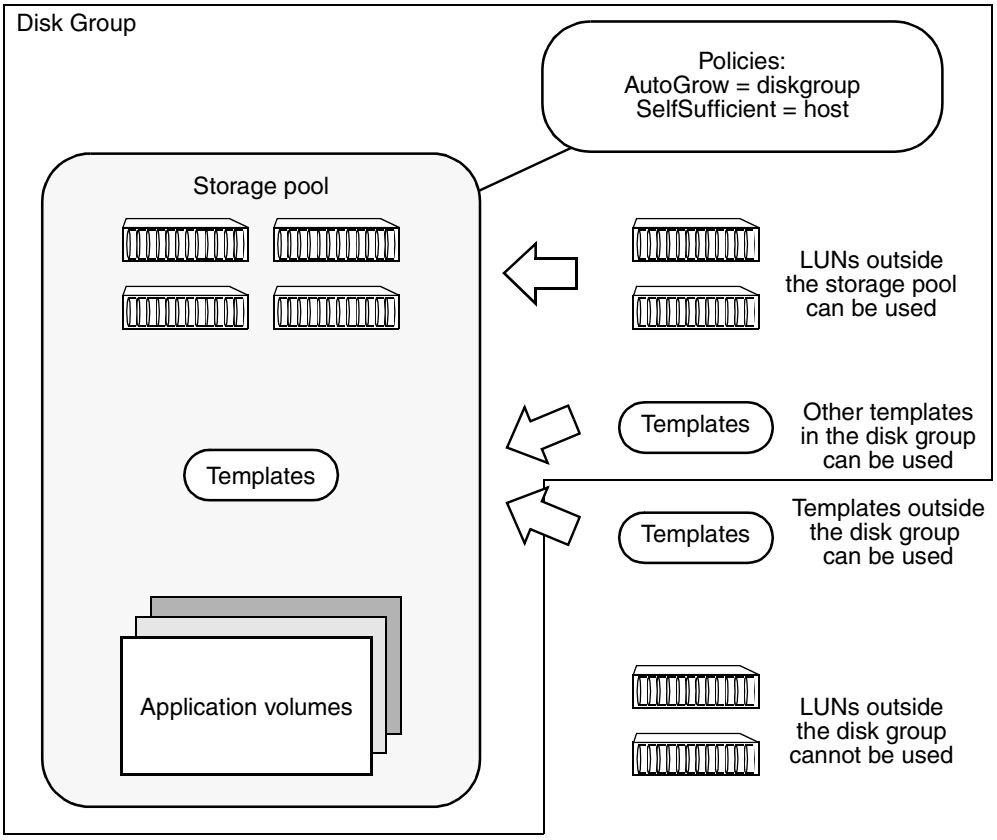
Figure 1-7 Effect of setting the value of the AutoGrow policy to pool



If the `AutoGrow` policy level were to remain set to the default value of `diskgroup`, this would allow the ISP to aggregate LUNs from the disk group.

Figure 1-8 illustrates how setting the `AutoGrow` policy to `diskgroup` allows LUNs from the same disk group that are outside the storage pool to be used, and setting the `SelfSufficient` policy to `host` allows templates that provide the required capabilities to be imported from outside the storage pool or disk group.

Figure 1-8 Effect of setting non-default storage pool policies

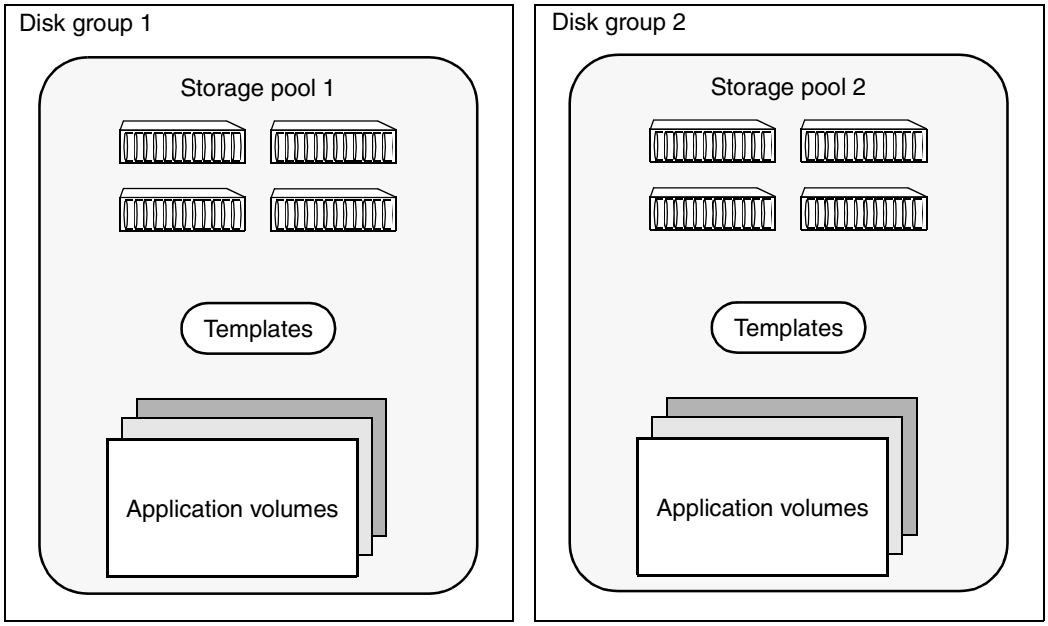


Note: Setting the storage pool policies away from their default values may not be desirable if you want to maintain close control over how a storage pool is used.

Arranging storage by volume usage

The simplest way to arrange storage so that its use is restricted to certain applications is to divide it between storage pools as shown in Figure 1-9. This requires that each storage pool is configured in a separate disk group. Each storage pool can have its own set of policies and templates that are tailored to the requirements of the volumes that are created from its LUNs.

Figure 1-9 Arranging storage by volume usage



Although this is the simplest way of arranging storage and is expected to be the most common, it may not provide sufficient flexibility for some installations.
See "Arranging storage by attributes" on page 32.

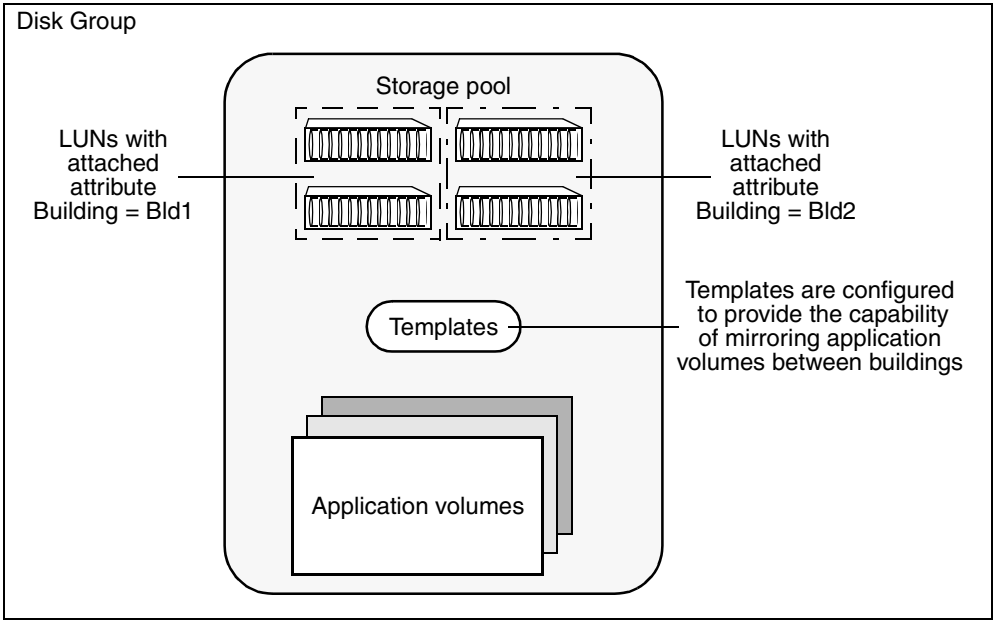
Arranging storage by attributes

You can use storage attributes to control how ISP assigns storage to application volumes. For example, you can use confinement rules to restrict some volumes to a subset of LUNs which share common attributes, such as caching to enhance I/O performance, or hardware RAID to provide redundancy and/or enhance performance.

As described in "About storage pool policies" on page 37, not all attributes of LUNs are capable of being discovered automatically. You can use disk tags, administered using the `vxdisk` command or the VEA graphical user interface, to manually attach such attributes to storage.

An example of using attached attributes is shown in Figure 1-10 where the templates can use the value of the `Building` attribute to provide the availability capability of mirroring volumes between different locations at a site.

Figure 1-10 Example usage of attached attributes



Alternatively, you could specify appropriate separation rules to instruct ISP to mirror volumes between buildings.

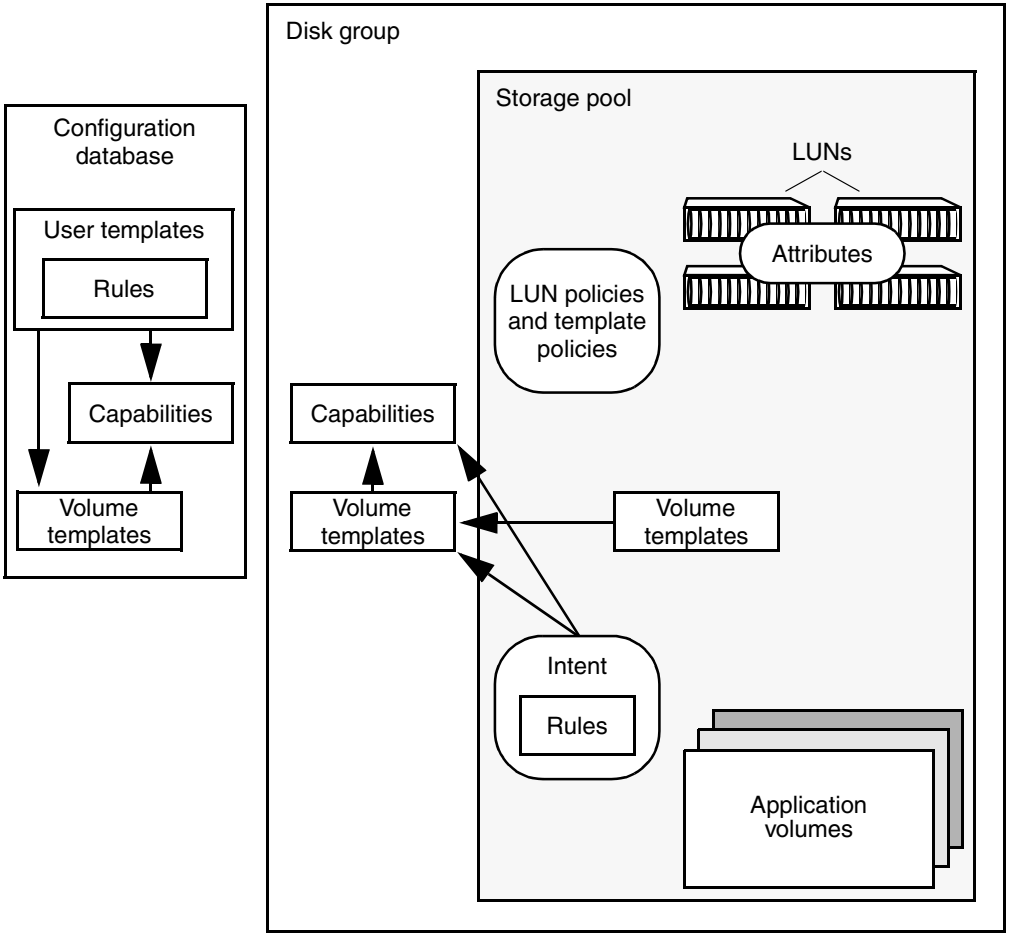
Another example of using attached attributes would be to tag certain LUNs within a storage pool as having the best performance. You could then use

confinement rules to ensure that certain volumes are only configured from this storage, while the remaining storage is used for other volumes with less critical performance requirements.

About ISP concepts

Figure 1-11 illustrates the dependencies between the various ISP concepts.

Figure 1-11 Relationship between ISP concepts



These concepts are defined in the following sections (in order of increasing abstraction):

About disk groups

A disk group is a named collection of disks that share a common configuration. Volumes and other VxVM objects must be created within a disk group, and are restricted to using disks from within that disk group.

About LUNs

A LUN, or *logical unit*, can either correspond to a single physical disk, or to a collection of disks that are exported as a single logical entity, or virtual disk, by a device driver or by an intelligent disk array's hardware. VxVM and other Veritas software modules may be capable of automatically discovering the special characteristics of LUNs, or you can use disk tags to define new storage attributes. Disk tags are administered by using the `vxdisk` command or the VEA graphical user interface.

About attributes

A storage attribute allows the properties of a LUN to be defined in an arbitrary conceptual space. For example, attributes can describe properties such as:

- Disk access name
- Disk media name
- Manufacturer
- Model type
- Physical location, such as rack number, frame number, floor, building, or site
- Hardware RAID configuration
- Failover properties
- Performance properties

You can use disk tags to create storage attributes in addition to those that are intrinsically associated with the disk hardware, and which are automatically discovered or assigned by VxVM. Disk tags are administered by using the `vxdisk` command or the VEA graphical user interface. For example, the `vxdisk settag` command can be used to assign tags and optional values to each disk:

```
# vxdisk -g mydg settag Room=room1 mydg01 mydg02 mydg03 mydg04
# vxdisk -g mydg settag Room=room2 mydg05 mydg06 mydg07 mydg08
```

This sets the attribute tag `Room` on the disks (`mydg01` through `mydg08`) with values that represent the physical location (`room1` or `room2`).

Attributes may be used to capture information about special features that storage possesses, such as:

- Hardware-supported cloning, such as EMC Business Continuity Volumes (BCV)
- Hardware-supported replication, such as the EMC Symmetrix Remote Data Facility (SRDF)
- Hardware redundancy, such as mirrored parity, and caching

It should only be necessary to enter such information manually if VxVM cannot discover it automatically. An example of a user-defined attribute is physical location.

Note: Attribute names and their string values are case sensitive. You can use the `vxdisk listtag` command or the VEA graphical user interface to discover the correct spelling of LUN attribute names.

About storage pools

A storage pool is defined within a disk group in VxVM for use by ISP. A storage pool is a policy-based container for LUNs and volumes. This means that the templates, capabilities and policies that are associated with a storage pool define how storage is organized within the pool.

Two types of storage pool are defined: *data pools* and *clone pools*.

For convenience, storage pool definitions are provided that include a number of associated templates that can be used for different purposes.

See "Storage pool" on page 191.

About data pools

A data storage pool is the first storage pool that is created within a disk group. All other storage pools that are subsequently created within a disk group are clone pools.

Note: Only one data pool can be created within a disk group.

It should not usually be necessary to move a data pool to another disk group. However, if this is required, only an entire data pool can be moved. An individual application volume within a data pool cannot be moved. If you want to move a data pool into a different disk group, you must ensure that the data pool contains only those objects that you require.

About clone pools

A clone storage pool contains one or more full-sized instant volume snapshot replicas of volumes within a data pool. (A volume snapshot is an image of a volume's contents at the moment in time that the snapshot was taken. See the *Veritas Volume Manager Administrator's Guide* for more information about volume snapshots.)

Note: Full-sized, linked break-off and space-optimized instant snapshots are supported for use with ISP. Third-mirror break-off volume snapshots are not supported.

In release 5.0 of VxVM, the introduction of linked break-off snapshot volumes means that it is unnecessary to set up a clone pool. A linked break-off snapshot volume can have different layout and characteristics from its parent volume, it can be set up in a data pool in a different disk group, and it retains its identity if it is relinked to its parent volume and subsequently broken off again.

A clone pool can be moved into another disk group, but an individual snapshot within it cannot. If you want to move a clone pool into a different disk group, you must ensure that the clone pool contains only those snapshots that you require. As an alternative, consider using linked break-off snapshots, as these can be created in a different disk group and storage pool from their parent volume.

About storage pool sets

A storage pool set is a bundled definition of the capabilities of a data pool and its clone pools. For convenience, you can use a storage pool set definition to define both pools in a single operation.

See "Storage pool set" on page 195.

About storage pool policies

A storage pool's policies define how it behaves when more storage is required, and when you try to create volumes whose capabilities are not permitted by the current templates. The two policies that are associated with a storage pool are:

`AutoGrow` Defines how a storage pool uses LUNs. This policy can take the following *level* values:

- `pool` Use storage that has been manually assigned to the storage pool.
- `diskgroup` Use storage that has been assigned to the disk group.

`SelfSufficient` Defines how a storage pool uses templates. This policy can take the following *level* values:

- `pool` Use templates that have been manually assigned to the storage pool.
- `diskgroup` Use templates that have been manually assigned to the disk group.
- `host` Use any template that has been installed in the ISP configuration database on this host.

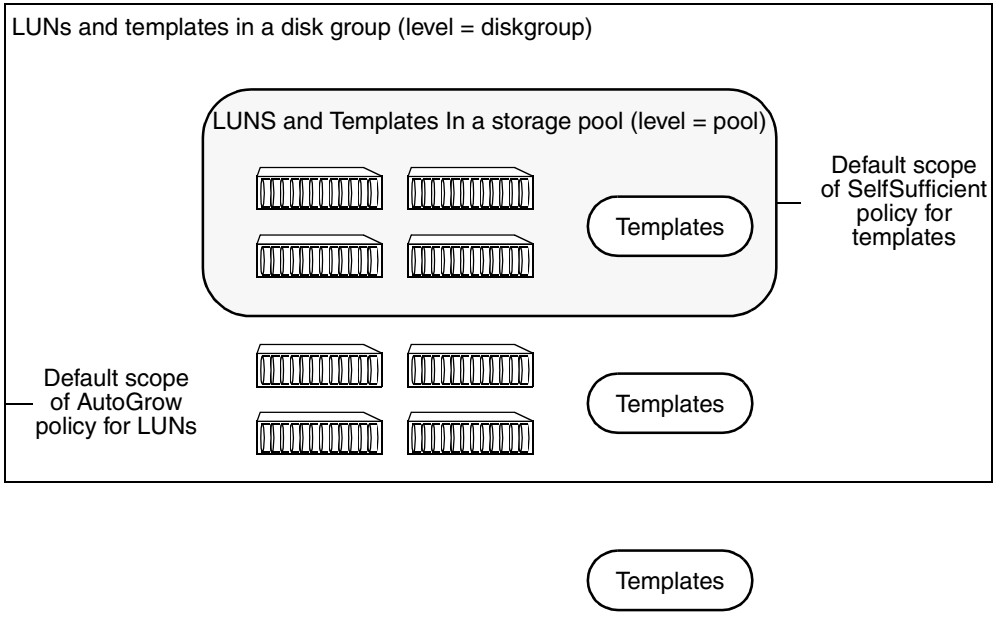
The values of these two policies can be combined to suit how the storage pool is to be used.

The policy levels that are associated with a storage pool control how it manages its templates and LUNs:

- The default level of the `AutoGrow` policy is `diskgroup`. This allows LUNs that are associated with the storage pool and its disk group to be used for allocating storage to volumes.
- For storage pools that are created explicitly by using the `vxpool` command, the default level of the `SelfSufficient` policy is `pool`. This allows only templates that are associated with the storage pool to be used for allocating storage to volumes. The storage pool can contain only volumes with the reliability and performance capabilities that are supported by these templates.
- For storage pools that are created implicitly by specifying the `-o intent` option to the `vxassist make` command when creating a volume, the default level of the `SelfSufficient` policy is `host`. This allows any template that has been installed in the ISP configuration database to be used.

Figure 1-12 illustrates the scope of the levels that can be configured as policy values for a storage pool.

Figure 1-12 Scope levels for storage pool policies



Note: LUNs in other storage pools or disk groups or outside any disk group are not immediately available for aggregating into another storage pool or disk group.

About capabilities

A capability is a feature that is provided by a volume. For example, a volume may exhibit capabilities such as performance and reliability to various degrees. Each type of capability is defined as a set of rules.

The following table shows some simple examples of capabilities that might be supported by a storage pool:

Capability	Adjustable parameters	Description
`DataMirroring`	`nmirs` — number of mirrors (plexes)	Provides reliability by using mirrored plexes.
`Raid5Capability`	`ncols` — minimum number of columns `nmaxcols` — maximum number of columns	Provides reliability by using RAID-5 parity.
`Raid5LogMirroring`	`nlogs` — number of log copies	Provides reliability for RAID-5 logs by mirroring.
`Striping`	`ncols` — minimum number of columns `nmaxcols` — maximum number of columns	Provides performance by striping across columns.

Capabilities have variable parameters that you can specify, such as the number of mirrors or columns. You can adjust the values of these parameters to tune the characteristics of a volume. If you do not specify parameter values, default values are used.

See “Capability” on page 185 for details of the capabilities that are provided.

About performance capabilities

Performance specifies the capabilities of a volume based on factors such as number of columns in striped volumes, stripe unit size, preferred characteristics of storage such as using prefabricated disks that are configured in hardware.

About reliability capabilities

Reliability specifies the level of redundancy that is required from a volume as a capability. Very reliable volumes have a high degree of redundancy. For example, a very reliable volume could be configured as a software mirrored volume built upon underlying prefabricated RAID-5 capable LUNs that are configured in the enclosure hardware. A medium-level redundant volume could be a simple 2-way mirror or RAID-5 volume configured either in software or on a suitable LUN within a single enclosure.

About rules

A rule is a statement written in the Veritas ISP language that specifies how a volume is to be created. A rule can define selection of storage or layout of storage. Rules are usually gathered together as templates for creating volumes, rather than being specified individually.

About storage selection rules

Storage selection rules specify what storage can be used to create volumes. The following are examples of storage selection rules:

Rule	Description
`affinity`	Attempts to select storage with shared attributes.
`confineto`	Restricts selection of storage by specifying attributes.
`exclude`	Prevents storage with certain attributes from being selected.
`select`	Specifies the storage to be used.
`separateby`	Defines fault domains for objects such as plexes.
`strong separateby`	Does not allow objects in different fault domains to share attributes.

For example, the following rule specifies that any LUNs may be selected from the named enclosures:

```
select "Enclosure"="ENC01", "Enclosure"="ENC02"
```

The next rule specifies that LUNs may be selected from the set of Hitachi disks that are located in `Room1`:

```
confineto "VendorName"="HITACHI", "Room"="Room1"
```

This example of a separation rule specifies that enclosures are to be treated as individual fault domains:

```
separateby "Enclosure"
```

About storage layout rules

Storage layout rules specify how storage is used to create volumes. The following are examples of storage layout rules:

Rule	Description
`log`	Specifies the type of log and its degree of redundancy.
`mirror`	Specifies how many mirrors a volume should have.
`parity`	Specifies whether redundancy is provided by using parity.
`stripe`	Specifies how many columns a volume should have.
`striped`	Specifies whether a volume is striped.

For example, the following rule specifies that a volume can be created using parity to provide data redundancy:

```
parity true
```

About volume templates

A volume template (or template for short) is a meaningful collection of rules that provide a capability. A template can specify one or more capabilities that a volume created using the template may have, and consists of a collection of storage selection and layout rules. For example, a template may allow you to create a volume that is able to tolerate the failure of a certain number of controllers, or that has a certain number of copies of the volume data.

When creating a volume, it is easier to specify its desired capabilities than to specify the precise layout of the volume on the available storage. ISP selects the appropriate templates and uses them to create a volume with the desired capabilities.

If you specify parameter values for a volume's capabilities, the rules that are defined within the chosen template use these values when selecting and laying out storage. If not specified, the default parameter values for a volume's capabilities are assumed by the template.

The following table shows some simple examples of templates and the capabilities that they might provide:

Template	Provides capabilities	Adjustable parameters for the capability
`DataMirroring`	`DataMirroring`	`nmirs` — number of mirrors (plexes)
`Raid5Volume`	`Raid5Capability`, `Raid5LogMirroring`	`ncols` — minimum number of columns `nlogs` — number of log copies `nmaxcols` — maximum number of columns
`Striping`	`Striping`	`ncols` — minimum number of columns `nmaxcols` — maximum number of columns

See "Volume template" on page 182.

About template sets

A template set consists of related capabilities and templates that have been collected together for convenience. Associating a template set with a storage pool is equivalent to associating each of its member templates separately with the storage pool.

See "Template set" on page 178.

About user templates

A user template (or user-defined template) defines an arbitrary collection of capabilities, templates and rules to which you want volumes of a certain type to conform. For example, you might want all volumes that you create to store database tables to share the same reliability and performance capabilities, and also that they only be allocated from a restricted set of storage.

It is useful to create user templates if you regularly create volumes with similar capabilities.

The following table shows some examples of user templates that might be created:

User template	Description
`OracleTable`	Provides a reliable high-performance volume that is suitable for use by a database table.
`OracleIndex`	Provides an extremely high-performance volume that is suitable for a database index.

As for templates, each user template can have a number of adjustable parameters that you can use for tuning the characteristics of the created volumes.

Note: A user template is not directly associated with a storage pool. Its association is implied by its reference to capabilities and templates.

About application volumes

An application volume is created by ISP, and then exported for use by an application such as a database or file system. It is identical to a traditional volume that you would create using `vxassist` and other VxVM commands, or via the VEA GUI.

Note: In this book, the term *volume* always means *application volume* unless it is specified that a traditional VxVM volume is meant.

About volume intent

The intent of a volume is a conceptualization of its purpose as defined by its characteristics and implemented by a template. ISP attempts to preserve the intent of a volume whenever the volume is reconfigured, resized, or relocated. Intent preservation automatically conserves capabilities such as reliability and performance, and observes additional rules such as allocating storage based on confinement and exclusion specifications.

Examples of using ISP from the command line

This section provides examples of using ISP from the command line and provides pointers to where more information can be found. You can also find summaries of the usage of commonly used commands in "Command summary" on page 205. Further examples may be found in "Examples of using ISP" on page 209 and in the manual page for each command.

Creating a data storage pool

Assuming that you have created a disk group, `mydg`, that contains several disks that you have initialized for use with VxVM, the following command creates a data storage pool, `mypool`, containing several disks, which supports the creation of striped-mirror volumes:

```
# vxpool -g mydg create mypool \
  dm=mydg01,mydg02,mydg03,mydg04,mydg04,mydg06 \
  pooldefinition=stripe_mirror_volumes
```

See "Creating a storage pool" on page 52.

Adding disks to a storage pool

You can use the `vxpool adddisk` command to add initialized disks to a storage pool. For example, this command adds two disks to the storage pool, `mypool`:

```
# vxpool -g mydg adddisk mypool dm=mydg07,mydg08
```

See "Adding disks to a storage pool" on page 55.

Creating an application volume

The following command creates a striped-mirror volume with three columns and two mirrors in the data storage pool, `mypool`, by specifying the capability `DataStripeMirror` together with the appropriate parameters:

```
# vxassist -g mydg -P mypool make stmrvol 2g \
  capability='DataStripeMirror(ncols=3,nmirs=2)' init=active
```

The `init=active` attribute makes the volume immediately available for use without performing any synchronization.

Having created the application volume, `stmrvol`, you can use the following command to prepare it for use with volume snapshots:

```
# vxassist -g mydg -P mypool addlog stmrvol nlog=2 logtype=dco
```

This command associates a data change object (DCO) and DCO volume with the volume. The attribute `nlog` is used to specify that the DCO volume has the same redundancy as the original volume. Alternatively, you can use the `vxsnap prepare` command to set up the volume for snapshots:

```
# vxsnap -g mydg prepare stmrvol ndcomirs=2
```

See “Creating application volumes” on page 61.

See “Creating instant snapshots” on page 101.

Resizing an application volume

If the application volume, `stmrvol`, that you have just created is not large enough for your needs, you can increase its size using this command:

```
# vxassist -g mydg -p mypool growto stmrvol 4g
```

Any file system or other application storage layout that you have created on the volume can be resized after you have grown the volume.

See “Resizing volumes online” on page 76.

Creating a clone storage pool

Assuming that you also want to create full-sized snapshots of application volumes in `mypool` that can be moved into a different disk group, the following command creates a clone storage pool, `myclpool`, in the same disk group as `mypool`:

```
# vxpool -g mydg create myclpool \
  dm=mydg09,mydg10,mydg11,mydg12 \
  autogrow=pool pooldefinition=mirrored_volumes
```

This pool supports the creation of mirrored volumes. It also has a non-default `autogrow` policy that prevents it aggregating disks from the disk group outside the pool.

See “Preparing storage pools for full-sized instant snapshots” on page 95.

Preparing a full-sized snapshot volume

Before you can take a full-sized snapshot of an application volume, you must prepare the empty volume that is to become the snapshot volume as shown here:

```
# vxassist -g mydg -P myclpool make mysnpvol 2g type=snapshot\
  init=active
```

This command automatically associates a data change object (DCO) and DCO volume with the volume.

See “Creating a volume for use as a full-sized instant or linked break-off snapshot” on page 96.

Taking a full-sized snapshot of an application volume

Having prepared an empty volume, you can take a snapshot of the application volume:

```
# vxsnap -g mydg make source=stmrvol/snapvol=mysnpvol/syncing=on
```

The following command starts a full synchronization of the snapshot volume, and blocks until this is complete:

```
# vxsnap -g mydg syncwait mysnpvol
```

See "Creating instant snapshots" on page 101.

Creating a cache volume for space-optimized snapshots

If you want to use space-optimized snapshots, you must prepare a storage cache where the snapshots can be created. This command sets up a 1GB cache volume, `mycache`, in the clone pool, `myclpool`:

```
# vxassist -g mydg -P myclpool make mycache 1g type=cachevolume
```

See "Creating a shared cache volume and preparing space-optimized snapshots" on page 97.

Preparing a space-optimized snapshot

Having created a cache volume, you now need to prepare the space-optimized snapshots that you require. This command sets up a space-optimized snapshot, `mysovol`, using the cache volume, `mycache`:

```
# vxassist -g mydg -P myclpool make mysovol 2g type=snapshot \
  cachevolume=mycache init=active
```

Note: The argument `2g` is the same as the length of the original source volume for which the snapshot is being prepared. This value defines the logical size of the snapshot. The actual amount of storage that the snapshot requires is less than this, and is limited by the size of the cache volume.

See "Creating a shared cache volume and preparing space-optimized snapshots" on page 97.

Taking a space-optimized snapshot of an application volume

Having prepared a cache volume and one or more empty space-optimized snapshots, you are ready to take a snapshot of the application volume:

```
# vxsnap -g mydg make source=stmrvol/snapvol=mysovol
```

See "Creating instant snapshots" on page 101.

Chapter 2

Storage pools

Veritas Intelligent Storage Provisioning (ISP) allows you to group storage with similar characteristics for creating volumes. A storage pool is a named collection of volumes and the LUNs with which they are associated.

The storage policies and rules that are associated with a storage pool determine its characteristics. Each storage pool represents a collection of volumes that are created according to these policies and rules. For example, a pool may be configured to allow allocation of LUNs from outside the pool, and to use only templates that are associated with the pool by specifying the appropriate `autogrow` and `selfsufficient` policies.

For more information on the rules that are supported for storage pools, refer to "Rules" on page 144.

This chapter describes how to use the `vxpool` utility to create and administer storage pools. For full information about this command, see the `vxpool`(1M) manual page.

Reserving and unreserving disks for use with ISP

When you initialize a disk for use with VxVM, for example, by running the `vxdiskadm` command, both ISP and non-ISP utilities have access to the space on the disk. You can reserve a disk for use with either set of utilities by setting the values of the `allocator_reserved` and `allocator_nouse` flags for the disk.

To reserve disks for exclusive use with ISP, and prevent non-ISP utilities from using these disks, select the menu item `22 Mark a disk as ISP-reserved for a disk group` from the main menu of the `vxdiskadm` command. Alternatively, you can use the following command to reserve a disk specified by its disk media name for use with ISP:

```
# vxedit [-g diskgroup] set "allocator_reserved=on" diskname
```

Note: A disk that is to be reserved for ISP must not contain any existing VxVM subdisks.

To remove the reservation flag from disks, select menu item `23 Turn off the allocator-reserved flag on a disk` from the main menu of the `vxdiskadm` command. Alternatively, you can use the following command to remove ISP reservation from a disk specified by its disk media name:

```
# vxedit [-g diskgroup] set "allocator_reserved=off" diskname
```

To prevent a disk from being used by ISP, enter the following command:

```
# vxedit [-g diskgroup] set "allocator_nouse=on" diskname
```

The following command removes the restriction on ISP using a disk:

```
# vxedit [-g diskgroup] set "allocator_nouse=off" diskname
```

Note: The flags `allocator_reserved` and `allocator_nouse` are mutually exclusive. Their values cannot both be set to `on` for a disk.

You can use the `vxdisk list` and `vxprint` commands to tell whether a disk has been reserved for ISP, as shown in the following examples:

```
# vxdisk list
DEVICE    TYPE          DISK      GROUP       STATUS
c0t8d0    auto:cdsdisk mydg1     mydg        online allocrsvd

# vxprint -g mydg mydg1
TYNAME    ASSOC     KSTATE  LENGTH    PLOFFS  STATE   TUTIL0 PUTIL0
dmmydg1   c0t8d0    -       35365968  -       ALLOC_RES  -      -
```

The `allocrsvd` status flag and the `ALLOC_RES` state indicate that a disk is reserved for use with ISP.

Organizing storage pools in a disk group

Before you can use ISP to create volumes in a disk group, you must first create any storage pools that you require in that disk group. A storage pool has associated disks, templates and policies. These policies control how the disks and templates are used when allocating storage from the pool to volumes.

The `vxpool organize` command simplifies the initial creation of one or more pools in a disk group by using a storage pool set definition. You can use this command to create a set of pools with policies and templates that are designed for a variety of different applications. A storage pool set consists of one data pool definition and one or more clone pool definitions. Each of these pool definitions typically consists of the pool type, the templates that the pool contains, and the pool policies.

For example, if you want your data volumes to be mirrored for redundancy, and your snapshot volumes to be striped for performance, you can choose a storage pool set definition where the data pool has associated templates that relate to mirroring, and the clone pool has associated templates that relate to striping. A suitable choice would be the `mirrored_data_striped_clones` storage pool set. To create these storage pools within a disk group, you would use the following command:

```
# vxpool -g diskgroup organize mirrored_data_striped_clones
```

See "Listing storage pool sets" on page 51.

See "Displaying storage pool set definitions" on page 51.

Listing storage pool sets

A storage pool set defines the storage pool types for a data storage pool and for the clone storage pools that are used to hold snapshots of the data storage pool's volumes.

To list all the available storage pool sets, use the following command:

```
# vxpool listpoolset
```

See "Storage pool set" on page 195.

Displaying storage pool set definitions

To display the definition of a storage pool set, use the following command;

```
# vxpool printpoolset storage_pool_set \
  [storage_pool_set ...]
```

For example, the following command displays the definition of the storage pool set, `mirrored_data_stripe_clones`:

```
# vxpool printpoolset mirrored_data_stripe_clones
storage_pool_set mirrored_data_striped_clones {
        description "The data volumes have multiple copies of
        data.
        Snapshot volumes have I/Os spread across multiple
        columns."
        descriptionid "{b84f1c64-1dd1-11b2-8b42-080020feef8b}",
        139
        display_name "Mirrored Data and Striped Snapshots"
        display_name_id "{b84f1c64-1dd1-11b2-8b42-
        080020feef8b}", 138
        data mirrored_volumes
        clone striped_volumes
};
```

See "Displaying storage pool definitions" on page 53.

Creating a storage pool

As a alternative to the `vxpool organize` command, you can use the `vxpool create` command to define and create a storage pool, as shown here:

```
# vxpool [-g diskgroup] create storage_pool
  [dm=dm1[,dm2...] \
  [description="description"] \
  [autogrow={1|pool}|{2|diskgroup}] \
  [selfsufficient={1|pool}|{2|diskgroup}|{3|host}] \
  [rules=rule [ rule ...]] \
  [pooldefinition=storage_pool_definition]
```

For example, the following command creates the storage pool, `mypool`, that contains disks `mydg02` and `mydg03`, and associates it with the disk group, `mydg`.

```
# vxpool -g mydg create mypool dm=mydg02,mydg03 \
  autogrow=diskgroup selfsufficient=pool
```

The `autogrow` policy level is set to `diskgroup` so the pool can use any storage within the disk group. The `selfsufficient` policy level of `pool` only allows the use of templates that have been manually assigned to the storage pool.

To simplify pool creation, you can also create a storage pool from a storage pool definition that is known to the system, as shown here:

```
# vxpool -g mydg create mypool pooldefinition=mirrored_volumes
```

Such definitions standardize storage pool policies and the templates that are to be associated with storage pools.

See “Listing available storage pool definitions” on page 53.

See “Displaying storage pool definitions” on page 53.

The pre-defined storage pool types include default policy values, and a set of volume templates that are installed. You can add disks to such storage pools.

See “Storage pool” on page 191.

See “Adding disks to a storage pool” on page 55.

Note: The disks that you assign to a storage pool must have already been initialized for use, and must belong to the disk group in which you are creating the storage pool.

The first storage pool that you create in a disk group is a data storage pool that contains application volumes. Any storage pools that you subsequently create in the disk group are clone storage pools that can be used to hold full-sized instant snapshots of the volumes in the data storage pool. You need only place such snapshots in a separate clone storage pool if they need to be created using different templates from their parent volumes, or if they are to be moved into a different disk group.

Listing available storage pool definitions

To list all the available storage pool definitions, use the following command:

```
# vxpool listpooldefinition
```

See "Storage pool" on page 191.

See "Displaying storage pool definitions" on page 53.

Displaying storage pool definitions

To display the details of one or more storage pool definitions, use the following command:

```
# vxpool printpooldefinition [storage_pool_definition] ...
```

If no storage pool definitions are named, the definitions for all storage pools are displayed.

For example, the following command displays the definition of the storage pool, `mirrored_volumes`:

```
# vxpool printpooldefinition mirrored_volumes
storage_pool mirrored_volumes {
        description "Volume has multiple copies of data."
        descriptionid "{b84f1c64-1dd1-11b2-8b42-080020feef8b}",
        117
        display_name "Data Mirroring"
        display_name_id "{b84f1c64-1dd1-11b2-8b42-
        080020feef8b}", 116
        volume_templates ArrayProductId,
        ConfineLogsToSimilarStorage,
        ConfineMirrorsToSimilarStorage, ConfineToSimilarStorage,
        ConfineToSpecificStorage, DCOLogMirroring,
        DCOLogStriping,
        DataMirroring, ExcludeSpecificStorage,
        InstantSnapshottable, LogsOnSeparateComponents,
        MirrorsOnSeparateComponents,
        MultipathingThroughMultiplePaths
        autogrow 2
        selfsufficient 1
};
```

Using disk group split and join with storage pools

Storage pools form the smallest unit that can participate in disk group split and join operations (see the section “Reorganizing the Contents of Disk Groups” in the “Creating and Administering Disk Groups” chapter of the *Veritas Volume Manager Administrator’s Guide* for details).

Note: Only clone storage pools can be moved between disk groups; data pools must remain in the disk group in which they were created.

The following command is used to split one or more clone pools from a source disk group to a newly-created target disk group:

```
# vxdg split sourcedg targetdg clonepool ...
```

All volumes (including instant snapshots) within the specified clone pools are moved to clone pools within the target disk group.

Note: Any disk group that contains a clone pool must also contain a data pool. If a clone pool is split from a disk group, an empty data pool is also created in the newly-created target disk group.

The following command is used to join two disk groups by merging the contents of the source disk group with the target disk group and then removing the source disk group:

```
# vxdg join sourcedg targetdg
```

All volumes (including full-sized instant snapshots) within the clone pool in the source disk group are moved to a clone pool of the same name within the target disk group.

Note: When two disk groups containing storage pools are joined, only one of the data pools may contain any disks, and the set of templates that are associated with one data pool must be a subset (including the null set) of the templates that are associated with the other data pool. The join operation fails if both data pools contain disks, if the sets of templates are overlapping or disjoint. At the end of the join operation, the data pool that contains the most templates is retained and the other data pool is deleted.

You can use the following command to move a clone pool between disk groups:

```
# vxdg move sourcedg targetdg clonepool ...
```

As for the split operation, a data pool is created in the target disk group if no data pool currently exists there.

Adding disks to a storage pool

To add one or more initialized disks to the storage pool, use the following command:

```
# vxpool [-g diskgroup] adddisk storage_pool \
  dm=dm1[,dm2,...]
```

The following example shows two disks, `mydg04` and `mydg05`, being added to the storage pool, `mypool`:

```
# vxpool -g mydg adddisk mypool dm=mydg04,mydg05
```

Note: Any disks that you add to a storage pool must have already been initialized for use, and must belong to the same disk group as the storage pool.

If a storage pool's autogrow policy is set to `diskgroup`, ISP automatically brings additional disks from the disk group into the storage pool as required.

Removing disks from a storage pool

To remove one or more disks from the storage pool, use the following command:

```
# vxpool [-g diskgroup] rmdisk storage_pool dm=dm1[,dm2...]
```

In the following example, the disks, `mydg01` and `mydg04`, are removed from the storage pool, `ReliablePool`, for use elsewhere:

```
# vxpool -g mydg rmdisk ReliablePool dm=mydg01,mydg04
```

Note: You cannot remove a disk from a storage pool if any volumes are configured on that disk. To remove such a disk, first use the `vxassist evacute` command to move the volumes away from the disk. When you have done this, you can then remove the disk from the storage pool.

Associating templates with a storage pool

To associate one or more volume templates with a storage pool, use the `vxpool assoctemplate` command as shown here:

```
# vxpool [-g diskgroup] assoctemplate storage_pool \
    template=t1[,t2,...]
```

For example, the following command associates the templates `Mirroring` and `Striping` with the storage pool, `ReliablePool`:

```
# vxpool -g mydg assoctemplate ReliablePool \
    template=Mirroring,Striping
```

Note: If the value of the `selfsufficient` policy is `diskgroup` or `host`, ISP first looks for storage that complies with the templates that are associated with the storage pool, and if no suitable templates are found, it then examines templates outside the pool.

See "Volume template" on page 182.

Associating template sets with a storage pool

To associate one or more template sets, including all the templates that they contain, with a storage pool, use the `vxpool assoctemplateset` command as shown here:

```
# vxpool [-g diskgroup] assoctemplateset storage_pool \
    template_set=ts1[,ts2,...]
```

For example, the following command associates the template set, `DataMirroring`, with the storage pool, `ReliablePool`:

```
# vxpool -g mydg assoctemplateset ReliablePool \
    template_set=DataMirroring
```

Note: If the value of the `selfsufficient` policy is `diskgroup` or `host`, ISP first looks for storage that complies with the templates that are associated with the storage pool, and if no suitable templates are found, it then examines templates outside the pool.

See "Template set" on page 178.

Dissociating templates from a storage pool

To dissociate a template from a storage pool, use the following command:

```
# vxpool [-g diskgroup] distemplate storage_pool \
    template=t1[,t2,...]
```

In the following example, the `PrefabricatedRaid5` template is disassociated from the storage pool, `ReliablePool`:

```
# vxpool -g mydg distemplate ReliablePool \
    template=PrefabricatedRaid5
```

After a template has been dissociated, ISP does not refer to the rules and capabilities in that template unless this is permitted by the `selfsufficient` policy that is set on a storage pool.

Displaying information about storage pools

To display information about one or more storage pools, use the following command:

```
# vxpool [-g diskgroup] print storage_pool \
  [storage_pool ...]
```

For example, the following command displays information about the storage pool, ReliablePool:

```
# vxpool -g mydg print ReliablePool
TY  NAME           AUTOGROW     SELFSUFFICIENT      DESCRIPTION
st  ReliablePool   2            2                   -
dm  disk04         -            -                   -
dm  disk02         -            -                   -
dm  disk01         -            -                   -
```

The command displays the policies and description for each specified storage pool, together with a list of the disks that are associated with the storage pool.

Note: The vxpool print command displays only disks that are in a pool, and which have at least one path available. Use the vxprint command to list full information about disks and their states.

Displaying storage pool policies

To display the policies that are set on a storage pool, use the vxpool getpolicy command as shown here:

```
# vxpool [-g diskgroup] pool getpolicy storage_pool
```

In the following example, this command is used to display the policies for the storage pool, ReliablePool:

```
# vxpool -g mydg pool getpolicy ReliablePool
Autogrow      Selfsufficient
2             1
```

Changing the policies on a storage pool

By default, ISP sets the values of the autogrow policy to diskgroup, and the selfsufficient policy to pool. To change the policies that are set on a storage pool, use the vxpool setpolicy command:

```
# vxpool [-g diskgroup] setpolicy storage_pool \
    [autogrow={1|pool}|{2|diskgroup}] \
    [selfsufficient={1|pool}|{2|diskgroup}|{3|host}]
```

The existing pool policies are overwritten by the new levels that you set.

In the following example, the policies on the storage pool, mypool, are changed to disallow the use of storage resources from outside the storage pool, and to allow all available templates to be considered for use when creating volumes:

```
# vxpool -g mydg setpolicy mypool autogrow=pool \
    selfsufficient=host
```

Listing storage pools within a disk group

To display a list of all storage pools within a disk group, use the vxpool list command as shown here:

```
# vxpool [-g diskgroup] list
```

For example, the following command lists all the storage pools within the disk group, mydg:

```
# vxpool -g mydg list
```

Finding storage pool sets containing specified pool definitions

To list the storage pool set definitions in the ISP Configuration Database that contain certain storage pool definitions, use the following command:

```
# vxpool -C listpoolset [pooldefn=pd1[,pd2...]]
```

For example, to list all the storage pool sets that contain the mirrored_volumes pool definition:

```
# vxpool -C listpoolset pooldefn=mirrored_volumes
TY NAME
ps mirrored_prefab_raid5_data_mirrored_clones
ps mirrored_data_striped_clones
```

If no storage pool sets are specified, the command lists all storage pool sets that are defined in the ISP Configuration Database.

Renaming a storage pool

To rename a storage pool, use the following command:

```
# vxpool [-g diskgroup] rename storage_pool new_name
```

For example, to rename the storage pool, mypool, as ReliablePool, you would use the following command:

```
# vxpool -g mydg rename mypool ReliablePool
```

Deleting a storage pool

To delete a storage pool, use the following command:

```
# vxpool [-g diskgroup] [-r] delete storage_pool
```

The `-r` option must be specified to dissociate all disks from the storage pool and then delete the storage pool as shown in this example:

```
# vxpool -g mydg -r delete ReliablePool
```

Note: If any volumes are present in the storage pool, you must delete these before the storage pool can be deleted.

Chapter 3

Creating application volumes

Volumes created by Veritas Intelligent Storage Provisioning (ISP) are similar to traditional non-ISP volumes that you create using the `vxassist` utility, but have the advantage that their intent is preserved and cannot accidentally be degraded. Volumes that are created by ISP can be managed by using commands such as `vxassist` or `vxvoladm`, or by using the VEA client graphical user interface. For information on using VEA to create volumes, refer to the VEA online help. This chapter describes how to use the `vxassist` command to create ISP volumes.

Note: To create application volumes successfully, the appropriate licenses must be present on your system. For example, you need a full Veritas Volume Manager license and a Veritas FlashSnap™ or FastResync license to use the instant snapshot feature. Vendors of disk arrays may also provide capabilities that require special licenses for certain features of their hardware.

Overview of the command line interface

You can use the `vxassist` command to create and manage ISP volumes provided that you have set up a storage pool in the disk group. ISP optimally assigns storage resources as defined and constrained by any parameter values, rules, capabilities and templates that you specify as arguments to the command. Capabilities provide the highest, most abstract way of specifying volumes. Rules provide the lowest, most direct means of specification. This gives you great freedom to create volumes that meet your requirements.

The `vxassist` command takes the general form:

```
# vxassist [options] keyword volume [additional_arguments] \
    [storage_specification] [attribute=value ...]
```

The *keyword* denotes the action that vxassist is to perform on the named volume. The storage specification defines the storage that can or cannot be used with an operation. This consists of a comma-separated list of disk media names and other storage attributes, such as Controller:*controller_name* to indicate all disks on a controller. Excluded storage is indicated by a ! prefix. Finally, attributes and their values can be used to specify further constraints on the operation.

Each invocation of vxassist is applied to only a single storage pool that has been configured within a disk group. The default disk group is that aliased by the setting of defaultdg. You can specify an alternate disk group by using the -g *diskgroup* option.

Note: Refer to the vxassist(1M) manual page for full details on using the vxassist command.

All operations in this chapter that use the vxassist command have an equivalent vxvoladm command, which is obtained by substituting vxvoladm for vxassist. Any other arguments to the command remain the same.

Setting default values for ISP volumes

You can define default values for ISP volumes in the file /etc/default/allocator or in an alternate file that you specify as an argument to the -d option. The defaults listed in this file are used unless they are overridden by a value specified on the command line. If a value is not defined in a defaults file or on the command line, vxassist uses a built-in default value.

Note: The file, /etc/default/allocator_readme, contains a copy of the defaults file as it is shipped and first installed.

By default, the attribute settings in the installed /etc/default/allocator file are commented out. If required, you can uncomment the entries, and edit their values. If you do this, you should first make a backup copy of the original unedited file to keep for reference.

The following entry for default_rules in the /etc/default/allocator file is commented out by default:

```
# default_rules=desired confineto "ProductId"
```

If enabled, this rule changes the default behavior of ISP so that it attempts to confine volumes to disks with the same product ID. The rule may also prevent

hot-relocation or volume transformation taking place if disks with the same product ID are not available.

For a single invocation of the `vxassist` command, you can override the default values of attributes that are defined in the `/etc/default/allocator` file, or that are built into `vxassist`, by specifying them as comma-separated arguments to the `-o` option.

Determining the maximum volume size

Before creating a volume, you may wish to check whether there is sufficient storage available. You can use the following command to determine the maximum size of a volume with a given capability that you can create in the data storage pool of a disk group:

```
# vxassist [-g diskgroup] [-P storage_pool] [-b] maxsize \
[storage_specification] [attribute=value ...]
```

Specify the `-b` option if you want to run this command in the background. Otherwise, it will block until it has completed its calculation of the maximum volume size.

Note: The storage that is available for allocation is based on the setting of the `AutoGrow` policy on a storage pool. If the value of this policy is set to `diskgroup` and not to `pool`, all suitable disks in the disk group are used in the calculation. You can use the `-P` option to specify that only storage from the specified pool is to be considered.

For example, the following command returns the maximum possible size of a mirrored volume with 3 mirrors that can be created from all the available free storage:

```
# vxassist -g mydg -P mypool maxsize layout=mirror nmir=3
```

This is the equivalent command if the `DataMirroring` capability is specified:

```
# vxassist -g mydg -P mypool maxsize \
  capability='DataMirroring(nmirs=3)'
```

The next example checks for the maximum volume size that can be created using prefabricated mirrors of a specified vendor type:

```
# vxassist -g peakdg -P peakpool maxsize \
  volume_template=PrefabricatedDataMirroring \
  rules='confineto "VendorName"="ACME"'
```

Note: The maximum size of a VxVM volume that you can create is 256TB.

Creating volumes

To create a volume with a specified length, specify the `make` keyword to `vxassist` as shown here:

```
# vxassist [options] make volume length \
  [storage_specification] [attribute=value ...]
```

It is possible to specify a combination of capabilities, templates and rules to this command. Recall that:

- A rule specifies a criterion for choosing and allocating storage.
- A template is a meaningful collection of rules that define a capability.
- A capability is a high-level description of what a volume can do.

The properties of the volume that is created are further determined by any parameters and other attributes that you specify. Rules, capabilities and templates may be combined as follows:

- If you specify rules along with capabilities, ISP creates volumes that comply to the rules that are defined in the capabilities, and also to those rules that you specify as arguments to `vxassist`.
- If you specify rules along with templates, ISP creates volumes that comply to the rules in the templates, and also to those rules that are specified as arguments to `vxassist`.
- If you specify capabilities along with templates, ISP selects the appropriate templates from those specified that conform to the desired capabilities.
- If you specify rules, capabilities and templates, ISP selects the appropriate templates from those specified that conform to the desired capabilities, and that adhere to the specified rules in addition to the rules in those templates.

See "Using capabilities, templates and rules" on page 137.

You can create volumes by specifying user templates that you have set up with the required redundancy, fault tolerance, or performance capabilities for the applications that you run at your site.

See "Creating and modifying user templates" on page 131.

Finally, in release 5.0 of VxVM, it is possible to use the `vxassist` command to create ISP volumes. If you are experienced in using specification attributes with `vxassist` to create volumes, or you have a number of legacy scripts that call `vxassist`, you may find it preferable to create ISP volumes in this way.

See "Creating volumes by using vxassist specification attributes" on page 65.

Creating volumes by using vxassist specification attributes

In release 5.0 of VxVM, it is possible to use the `vxassist` command to create ISP volumes (for which intent is preserved) in addition to non-ISP volumes (for which intent has no meaning). The `vxassist` command now accepts the same specification of templates, capabilities and rules as the `vxvoladm` command, and a set of `vxassist` storage specification attributes are automatically translated into equivalent ISP rules.

See "Equivalences between vxassist attributes and ISP rules" on page 201.

If the `-o intent` option is specified, `vxassist` creates an ISP volume, and it also sets up a storage pool in the disk group if one does not already exist. If a storage pool already exists in a disk group, the vxassist command attempts to create an ISP volume unless you specify the `-o nointent` option.

The following are examples of using the `vxassist` command in this way. Subsequent sections describe how to use capabilities, templates and rules to create ISP volumes with requirements that are beyond the scope of traditional `vxassist` specification attributes.

Creating a mirrored volume

The following command creates a 1-gigabyte volume with the default number of 2 mirrors:

```
# vxassist -g mydg -o intent make mir2vol 1g layout=mirror \
  nmir=2 init=active
```

The `init=active` attribute makes the volume immediately available for use without attempting to synchronize its empty plexes. The `-o intent` option is used to specify that an ISP volume is required, and to create a storage pool in the disk group if one does not already exist.

The following command creates a 1-gigabyte volume with 3 mirrors:

```
# vxassist -g mydg -P mypool make mir3vol 1g layout=mirror \
  nmir=3 init=active
```

In this example, the `-o intent` option is implicit as the `-P` option is used to specify the name of the storage pool that is to be used.

Creating a mirrored volume with mirrors on separate enclosures

The following command creates a 2-gigabyte mirrored volume with 2 mirrors, and with the mirrors located on separate enclosures:

```
# vxassist -g mydg -P mypool make strpvol 2g layout=mirror \
  nmir=2 mirror=enclosure
```

Such a volume tolerates the failure of one enclosure and provides greater reliability.

Creating a striped volume

The following command creates a 2-gigabyte striped volume with 10 columns:

```
# vxassist -g mydg -P mypool make strpvol 2g layout=stripe \
  ncols=10
```

Creating a mirrored-stripe volume with mirrors on separate controllers

The following command creates a 2-gigabyte mirrored-stripe volume with 8 columns, and with the mirrors located on separate controllers:

```
# vxassist -g mydg -P mypool make strpvol 2g \
  layout=mirror-stripe ncols=8 mirror=controller
```

Creating a RAID-5 volume with a RAID-5 log

The following command creates a 2-gigabyte RAID-5 volume with 8 data columns and a RAID-5 log that has 2 redundant copies:

```
# vxassist -g mydg -P mypool make r5vol 2g layout=raid5 \
  ncols=8 nraid5logs=2
```

Creating volumes by specifying capabilities

A capability is a high-level description of a volume, for example, `DataMirroring`, `Striping` or `PrefabricatedRaid5`. ISP automatically selects a suitable template from those that provide the desired capability. As such, this is the most abstract way of creating volumes using `vxassist` as it requires the least specification by you. You can customize a capability by specifying values for any variable parameters that it defines.

For a list of predefined capabilities that are supported, see “Capability” on page 185. The following sections provide some examples of creating volumes with these capabilities.

Creating a mirrored volume

The following command creates a 1-gigabyte volume with the default number of 2 mirrors:

```
# vxassist -g mydg -P mypool make mir2vol 1g \
  capability='DataMirroring' init=active
```

The `init=active` attribute makes the volume immediately available for use without attempting to synchronize its empty plexes.

The following command creates a 1-gigabyte volume with 3 mirrors:

```
# vxassist -g mydg -P mypool make mir3vol 1g \
  capability='DataMirroring(nmirs=3)' init=active
```

Creating a mirrored volume with mirrors on separate enclosures

The following command creates a 2-gigabyte mirrored volume with 2 mirrors, and with the mirrors located on separate enclosures:

```
# vxassist -g mydg -P mypool make strpvol 2g \
  capability='DataMirroring,MirrorsOnSeparateComponents'
```

Such a volume tolerates the failure of one enclosure and provides greater reliability. Such a capability can be combined with multipathing to provide resilience against the failure of one of the paths to an enclosure:

```
# vxassist -g mydg -P mypool make strpvol 2g \
  capability='DataMirroring,MirrorsOnSeparateComponents,\
MultipathingThroughMultiplePaths'
```

Creating a striped volume

The following command creates a 2-gigabyte striped volume with 10 columns:

```
# vxassist -g mydg -P mypool make strpvol 2g \
  capability='Striping(ncols=10)'
```

Creating a mirrored-stripe volume with mirrors on separate controllers

The following command creates a 2-gigabyte mirrored-stripe volume with 8 columns, and with the mirrors located on separate controllers:

```
# vxassist -g mydg -P mypool make strpvol 2g \
  capability='DataMirrorStripe(ncols=8),\
MirrorsOnSeparateComponents(component="Controller")'
```

Creating a striped-mirror volume from prefabricated mirrors

The following command creates a 10-gigabyte striped-mirror volume with 8 columns, and which uses prefabricated mirrors that are exported by an array to form each column:

```
# vxassist -g mydg -P mypool make strpvol 2g \
  capability='Striping(ncols=8),PrefabricatedDataMirroring'
```

Creating a RAID-5 volume with a RAID-5 log

The following command creates a 2-gigabyte RAID-5 volume with 8 data columns and a RAID-5 log that has 2 redundant copies:

```
# vxassist -g mydg -P mypool make r5vol 2g \
  capability='Raid5Capability(ncols=8),\
Raid5LogMirroring(nlogs=2)'
```

Creating volumes by specifying capabilities and rules

You can specify storage selection and layout rules in addition to capabilities when creating volumes. This allows you to select explicitly which storage can and cannot be used for certain volumes. For example, you might do this to ensure that volumes are created on storage with certain performance or reliability that is implemented in an array's hardware.

Refer to "Rules" on page 144 for details of the different types of rule and their usage.

Creating a RAID-5 volume on prefabricated storage

The following command creates a 10-gigabyte volume on a prefabricated RAID-5 disk that is exported from an array made by a specified vendor:

```
# vxassist -g mydg -P mypool make perfr5vol 10g \
  capability='PrefabricatedRaid5' \
  rules='confineto "VendorName"="ACME"'
```

Creating a mirrored volume on enclosures in separate locations

The following command creates a mirrored volume that tolerates the failure of a single enclosure, where each enclosure is in a different room. This avoids single point of failure and ensures greater reliability.

```
# vxassist -g mydg -P mypool make mirvol 10g \
  capability='DataMirroring' \
  rules='separateby "Room","Enclosure"'
```

In this example, it is assumed that the `vxdisk settag` command has been used to assign tags for the user-defined attribute `Room` to each disk, for example:

```
# vxdisk -g mydg settag Room=room1 mydg01 mydg02 mydg03 mydg04
# vxdisk -g mydg settag Room=room2 mydg05 mydg06 mydg07 mydg08
```

Creating a striped volume excluding certain disks

The following command creates a striped volume on storage other than that in `Room1` and certain disks in enclosure, `Enc1`:

```
# vxassist -g mydg -P mypool make strvol 1g \
  capability='Striping(ncols=4)' \
  rules='exclude allof("Room"="Room1","DeviceName"="Enc1_1",\
"DeviceName"="Enc1_2")'
```

Creating volumes by specifying templates

Volumes can be created by specifying templates instead of capabilities. The following example demonstrates how to create a mirrored volume using a template:

```
# vxassist -g mydg -P mypool make mirvol 1g \
  volume_template=DataMirroring
```

Note: If you specify templates when creating a volume, you must ensure that these templates are self-sufficient, and that they do not depend on templates that are not specified on the command line. This applies to all cases where templates are named by themselves, or in combination with a mixture of capabilities and rules.

A template can refer to other templates in the following ways:

- It can be an extension of another template.
- It can apply another template.
- It can require or inherit a capability that is provided by another template.

If one or more of these conditions apply, ISP may not be able to use the given templates and volume creation may fail. To view the details of a template, use the following command:

```
# vxtemplate [-g diskgroup] print template_name
```

Examine the definitions of the `extends`, `inherits`, `requires` and `apply` fields to see the other templates on which the specified template depends.

See "Volume templates" on page 140.

For example, the following command fails because of a template dependency:

```
# vxassist -g mydg -P mypool make myvol 1g \
  volume_template=PrefabricatedRaid5
VxVM vxassist ERROR V-61-49872-28 Template PrefabricatedRaid5 is
not valid for the operation. Either the template itself is
invalid or one or more of the related templates/capabilities are
not in the scope.
```

The `vxtemplate print` command is then run to examine the definition of the `PrefabricatedRaid5` volume template:

```
# vxtemplate -g mydg print PrefabricatedRaid5
volume_template PrefabricatedRaid5 {
    provides PrefabricatedRaid5
    rules {
        apply ArrayProductId
        confineto "Parity" ="1"
    }
};
```

The `apply` rule in the listing shows that the `PrefabricatedRaid5` template depends on the `ArrayProductId` template. If the `ArrayProductId` template does not depend on any other templates, the `vxassist make` command should be modified to list `ArrayProductId` in addition to `PrefabricatedRaid5`:

```
# vxassist -g mydg -P mypool make myvol 1g \
  volume_template=PrefabricatedRaid5,ArrayProductId
```

Creating volumes by specifying templates and rules

The following command creates a striped volume by using a template, and specifies a rule to ensure that storage is allocated only from a single enclosure in a specified location:

```
# vxassist -g mydg -P mypool make strvol 1g \
  volume_template='Striping' \
  rules='confineto eachof("Enclosure","Room"="Room2")'
```

Note: If you use a volume template to create a volume, and do not specify any capabilities, the values of all the variables that are used in the volume template are taken from the default values defined in the capability. An error is returned if such a variable does not have a default value.

Creating volumes by specifying templates and capabilities

If you specify templates in addition to capabilities when creating a volume, this restricts ISP to choosing from the specified templates.

The following example shows how to specify both capabilities and templates:

```
# vxassist [-g diskgroup] [-P storage_pool] make volume length \
  capability='capability[,capability...]' \
  volume_template=template[,template...]
```

If you specify rules in addition to capabilities and templates, these rules are applied after ISP has selected the templates that satisfy the required capabilities.

Creating volumes by specifying user templates

Once you have set them up, user templates provide the simplest way of creating volumes for use with the applications that you run. For example, you can configure user templates that ensure that volumes that you create to store database tables all share the same reliability and performance capabilities, and that they are allocated from a restricted set of storage. The following example shows how you might use a user template to create a volume for a database table:

```
# vxassist -g dbdg -P dgpool make Customers 15g \
  user_template=DBTable
```

Creating volumes with associated tags

Volume tags are used to implement the Dynamical Storage Tiering (DST) feature of the Storage Foundation software. For more information about this feature, see the *Veritas File System Administrator's Guide*.

To create a volume with an associated tag and optional tag value, specify the `tag` attribute as shown in this example:

```
# vxassist -g dbdg -P dgpool make products 1g \
  user_template=DBTable tag=db_table=Products
```

This creates a volume with a tag named `db_table`, which has the value `Products`.

Creating multiple volumes with the same prefix

To create multiple volumes with the same name prefix, specify a numeric argument to the `nvol` parameter, as shown in this example:

```
# vxassist -g mydg -P mypool make mirvol 15g \
  layout=mirror nmir=2 nvol=3
```

This creates three mirrored volumes named `mirvol1`, `mirvol2` and `mirvol3`. If volumes with the same name prefix and numeric suffix already exist, the numbering of the new volumes continues from the highest number found plus 1. For example, if there are existing volumes named `mirvol1` and `mirvol2`, the new volumes are named `mirvol3`, `mirvol4` and `mirvol5`.

Creating multiple volumes as a volume group

If you choose to create volumes individually, allocation may eventually fail when the available storage is exhausted. The -M option to the `vxassist` command allows you to create several volumes at the same time while making the most efficient use of the available storage resources. ISP automatically chooses the best way to allocate storage to the volumes. A set of multiple volumes that are created by this method is referred to as a *volume group*.

For convenience, it is easiest to define one or more volume groups in a definition file, and have `vxassist` read this file to create the volumes as shown here:

```
# vxassist -M make < filename
```

See "Volume group definition syntax" on page 197.

A sample definition might contain the following `volumegroup` entry:

```
volumegroup {
    diskgroup "mydg"
    rules {
        separateby "Enclosure"
        exclude "Enclosure"="ENC1"
```

```
        }
        volume "mirvol1" 10g {
            capability 'DataMirroring(nmirs=2)'
        }
        volume "mirvol2" 10g {
            capability 'DataMirroring(nmirs=2)'
        }
        volume "mirvol3" 10g {
            capability 'DataMirroring(nmirs=2)'
        }
    };
```

This specifies three 10-gigabyte mirrored volumes in the disk group, `mydg`, with the data mirrors placed on separate enclosures, but excluding enclosure, `ENC1`.

The next sample definition specifies a prefabricated striped volume on storage that is restricted to the location `Room1`, and a volume that is mirrored across controllers in the disk group `testdg`. In addition, the rules for the volume group specify that the volumes are to be created on separate enclosures, and that storage may only be allocated on disks from a given vendor.

```
    volumegroup {
        diskgroup "testdg"
        rules {
            separateby "Enclosure"
            confineto "VendorName"="ACME"
        }
        volume "strpvol" 10g {
            rules {
                confineto "Room=Room1"
            }
            capability 'PrefabricatedStriping'
        }
        volume "mirdvol" 1g {
            rules {
                separateby "Controller"
            }
            capability 'DataMirroring(nmirs=2)'
        }
    };
```

To create multiple volumes with the same name prefix, specify a numeric argument to the `nvol` parameter in the `volume` section, as shown in this example:

```
    volumegroup {
        diskgroup "mydg"
        rules {
            separateby "Enclosure"
            exclude "Enclosure"="ENC1"
        }
        volume "mirvol" 10g {
            nvol 3
            capability 'DataMirroring(nmirs=2)'
```

```
        }
    };
```

This creates three volumes named `mirvol1`, `mirvol2` and `mirvol3`. If volumes with the same name prefix and numeric suffix already exist, the numbering of the new volumes continues from the highest number found plus 1.

Creating a volume for use with snapshots and DRL

If the dirty region logging (DRL) or instant snapshot features are going to be used with an application volume, a version 20 data change object (DCO) and DCO volume must be associated with the volume.

If you want to create a volume of which you can take instant snapshots, the `InstantSnapshottable` capability or template must be specified (unless this capability is already implied by another capability or template):

```
# vxassist -g mydg -P mypool make mir3vol 1g \
  capability='DataMirroring(nmirs=3),InstantSnapshottable' \
  init=active
```

By default, a data change object (DCO) and DCO volume with a single plex are associated with a volume to enable the instant snapshot capability. To increase the redundancy of the DCO volume, specify the `DCOLogMirroring` capability as shown here:

```
# vxassist -g mydg -P mypool make mir3vol 1g \
  capability='DataMirroring(nmirs=3),InstantSnapshottable,\
  DCOLogMirroring(nlogs=3)' init=active
```

In this example, the same degree of redundancy is created for the DCO volume as for the data volume.

The DCO and DCO volume that are added by the `InstantSnapShottable` capability also provide the dirty region logging (DRL) capability that speeds recovery of mirrored volumes after a system crash. To enable DRL, specify either the attribute `drl=on` for full DRL, or `drl=sequential` for sequential DRL (used with volumes containing database replay logs). By default, DRL is disabled (`drl=off`). For example, the following command enables DRL for the volume, `mir3vol`, that is being created:

```
# vxassist -g mydg -P mypool make mir3vol 1g \
  capability='DataMirroring(nmirs=3),InstantSnapshottable' \
  drl=on init=active
```

Note: Traditional DRL logs are implemented using a dedicated plex for each log copy, rather than using a DCO volume, and are supported for mirrored volume layouts only.

See "Adding logs to a volume" on page 86.

Chapter 4

Administering application volumes

This chapter describes how to perform maintenance tasks on volumes that you have created using Veritas Intelligent Storage Provisioning (ISP). This includes resizing and changing the capabilities of volumes without taking them offline.

The operations in this chapter use the `vxassist` and `vxvoladm` commands. For full details, see the `vxassist`(1M) and `vxvoladm`(1M) manual pages.

Note: All operations in this chapter that use the `vxassist` command have an equivalent `vxvoladm` command, which is obtained by substituting `vxvoladm` for `vxassist`. Any other arguments to the command remain the same. The `vxassist` command does not support the `migrate`, `printintent`, `printrules`, `reversemigrate` and `reversemigrateall` operations of the `vxvoladm` command.

For details of how to display information about volumes, monitor and control tasks, and how to use the `vxresize` command to resize a volume and an underlying file system, see the "Administering Volumes" chapter of the *Veritas Volume Manager Administrator's Guide*.

Resizing volumes online

Increasing or decreasing the size of a volume is an operation that can be performed while a volume is online. The following `vxassist` operations are available for resizing a volume:

- `growto` — Increase volume size to a specified length.
- `growby` — Increase volume size by a specified length.
- `shrinkto` — Reduce volume size to a specified length.
- `shrinkby` — Reduce volume size by a specified length.

You can specify the length argument in sectors, kilobytes, megabytes, gigabytes or as a percentage by adding the unit of measure as a suffix (`s`, `m`, `k`, `g`, `t`, `p`, `e` or `%`) to the length value. If no unit is specified, sectors are assumed.

If the volume that is being resized has any linked mirror volumes, these mirror volumes are also resized in the same operation. However, any full-sized snapshots that have been broken-off are not resized, and new snapshots must be created. Space-optimized instant snapshots are not affected as they are not full-sized versions of their parent volumes.

See "Administering instant snapshots" on page 93.

Warning: If you use `vxassist` to resize a volume, do not shrink it below the size of the file system that is located on it. If you do not shrink the file system first, you risk unrecoverable data loss. If you have a VxFS file system, shrink the file system first, and then shrink the volume. Other file systems may require you to back up your data so that you can later recreate the file system and restore its data. Alternatively, you can use the `vxresize` command to resize both the volume and its file system where this is supported. See the `vxresize`(1M) manual page for more information.

Note: If you use the `vxassist` command to resize application volumes of type `fsgen` or `raid5`, you must specify the `-f` (force) option to the command. You must also specify the `-f` option if growing a volume would violate any rules.

Determining the maximum size of a volume

You can use the following command to determine by how much you can grow a volume using the available storage:

```
# vxassist [-g diskgroup] maxgrow volume [attributes...]
```

You can use storage specification attributes with this command to restrict the storage that is taken into consideration by the calculation. For example, the following command specifies that any JBOD storage may be used to resize volume, `vol02`:

```
# vxassist -g mydg maxgrow vol02 \
  use_storage='allof("Enclosure"="Disk")'
```

Increasing the size of a volume to a specified length

The following command grows a volume to a specified length:

```
# vxassist [-g diskgroup] [-f] growto volume length \
  [attributes...]
```

You can use storage specification attributes with this command to restrict the storage that is used to grow a volume. For example, the following command expands the volume `vol1` to 10 gigabytes, but excludes storage on controller `c1` from being allocated:

```
# vxassist -g mydg growto vol1 10g \
  use_storage='noneof("Controller"="c1")'
```

This operation fails if the new length specified is smaller that the current size of the volume.

Increasing the size of a volume by a specified amount

The following command grows a volume by a specified amount:

```
# vxassist [-g diskgroup] [-f] growby volume length \
  [attributes...]
```

For example, the following command grows the volume `vol1` by 1 gigabyte by allocating contiguous storage:

```
# vxassist -g mydg growby vol1 1g layout=contig
```

The setting `layout=contig` allows only contiguous regions of disk to be used for plexes or columns. By default, the length of a volume is increased by first extending existing subdisks in the volume if possible, and then by adding and associating new subdisks. This default behavior corresponds to the attribute setting `layout=nocontig`.

Reducing the size of a volume to a specified length

The following command reduces the length of a volume:

```
# vxassist [-g diskgroup] [-f] shrinkto volume length \
  [attributes...]
```

For example, the following command shrinks the volume `vol1` to 5 gigabytes:

```
# vxassist -g mydg shrinkto vol1 5g
```

This operation fails if the new length specified is larger that the current size of the volume.

Reducing the size of a volume by a specified amount

The following command reduces the length of a volume by a specified amount:

```
# vxassist [-g diskgroup] [-f] shrinkby volume length \
  [attributes...]
```

For example, the following command shrinks the volume `vol01` by 500 megabytes:

```
# vxassist -g mydg shrinkby vol1 500m
```

Growing and shrinking multiple volumes

If the `-M` option is specified to the `vxassist` command, multiple volumes can be resized in the same operation. For convenience, it is easiest to define the volumes that are to be resized in a file, and then have `vxassist` read this file to resize the volumes as shown in this example:

```
# vxassist -M growby < filename
```

Each line of input defines the disk group, volume and length parameters for the resize and has the following format:

```
diskgroup dgname volume volume_name length
```

When used in scripts, the input may conveniently be taken from a “here document” as shown in this example:

```
vxassist -M shrinkto <<!!
diskgroup mydg volume vol01 10g
diskgroup mydg volume vol02 12g
!!
```

Setting tags on volumes

Volume tags are used to implement the Dynamic Storage Tiering feature of the Storage Foundation software. For more information about this feature, see the *Veritas File System Administrator's Guide*.

You can use the following forms of the `vxassist` command to set a named tag and optional tag value on a volume, to replace a tag, and to remove a tag from a volume:

```
# vxassist [-g diskgroup] settag volume tagname[=tagvalue]
# vxassist [-g diskgroup] replacetag volume oldtag newtag
# vxassist [-g diskgroup] removetag volume tagname
```

To list the tags that are associated with a volume, use this command:

```
# vxassist [-g diskgroup] listtag volume
```

To list the volumes that have a specified tag name, use this command:

```
# vxassist [-g diskgroup] list tag=tagname volume
```

Tag names and tag values are case-sensitive character strings of up to 256 characters. Tag names can consist of letters (A through Z and a through z), numbers (0 through 9), dashes (-), underscores (_) or periods (.) from the ASCII character set. A tag name must start with either a letter or an underscore. Tag values can consist of any character from the ASCII character set with a decimal value from 32 through 127. If a tag value includes any spaces, quote the specification to protect it from the shell, as shown here:

```
# vxassist -g mydg settag myvol "dbvol=table space 1"
```

Dotted tag hierarchies are understood by the `list` operation. For example, the listing for `tag=a.b` includes all volumes that have tag names that start with `a.b`.

The tag names `site`, `udid` and `vdid` are reserved and should not be used. To avoid possible clashes with future product features, it is recommended that tag names do not start with any of the following strings: `asl`, `be`, `isp`, `nbu`, `sf`, `symc` or `vx`.

Preparing a volume for DRL and snapshot operations

To prepare an application volume for dirty region logging (DRL) and instant snapshot operations, a data change object (DCO) and DCO volume must be associated with the volume.

Note: The procedure in this section describes how to add a version 20 DCO and DCO volume to an application volume that you previously created in a disk group with a version number of 110 or greater. If you are creating a new volume in a disk group with a version number of 110 or greater, you can specify the co-creation of a DCO and DCO volume and enabling of DRL.
See "Creating a volume for use with snapshots and DRL" on page 73.
See "Adding logs to a volume" on page 86.

You may need an additional license to use the DRL and FastResync features.

Use the following command to add a version 20 DCO and DCO volume to an existing volume:

```
# vxassist [-g diskgroup] [-P storage_pool] addlog volume \
  [nlog=number] logtype=dco [regionsize=size] \
  [drl=yes|no|sequential]
```

The `nlog` attribute specifies the number of DCO plexes that are created in the DCO volume. It is recommended that you configure as many DCO plexes as there are data plexes in the volume. For example, specify `nlog=3` for a volume with 3 data plexes.

The value of the `regionsize` attribute specifies the size of the tracked regions in the volume. A write to a region is tracked by setting a bit in the change map. The default value is `64k` (64KB). A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization.

To enable DRL logging on the volume, specify `drl=yes`. If sequential DRL is required, specify `drl=sequential`.

You can also specify `vxassist`-style storage attributes to define the disks that can and/or cannot be used for the plexes of the DCO volume.

Removing support for DRL and snapshots from a volume

To remove support for DCO-based DRL and instant snapshots from a volume, use the following command to remove the DCO and DCO volume that are associated with the volume:

```
# vxassist [-g diskgroup] [-P storage_pool] removeall log \
  volume logtype=dco
```

Note: This command fails if the volume is part of a snapshot hierarchy.

Evacuating a volume

If you need to remove or disable a disk, you must first move any data off the disk by evacuating it. Similarly, if the whole or parts of a volume are configured on disks that are needed for other purposes, or which are not optimal or appropriate for use by the volume, you can evacuate these.

To evacuate certain disks on which a volume is configured, use either of the following commands to specify these:

```
# vxassist [-g diskgroup] [-b] evacuate disk volume \
  !dmname ... [use_storage=rule[,...]] [attributes...]

# vxassist [-g diskgroup] [-b] evacuate disk volume \
  evac_storage=rule[,...] [use_storage=rule[,...]] \
  [attributes...]
```

The disk media names of the disks that are to be evacuated are specified as `!`*dmname* where *dmname* is a disk media name. Alternatively, you can also use the storage specification attribute, `evac_storage`, to specify rules for evacuating currently used storage. The -b option may be specified to run the evacuation as a background task. ISP allocates suitable storage according to the `autogrow` policy of the storage pool, or you can use the storage specification attribute, `use_storage`, to specify rules for allocating new storage.

For example, the following command evacuates disks `mydg01` and `mydg02` on which volume, vol01, is configured, and specifies disks `mydg03` and `mydg04` as destination disks:

```
# vxassist -g mydg evacuate disk vol01 !mydg01 !mydg02 \
  use_storage='allof("DM"="mydg03","DM"="mydg04")'
```

Not only disks can be evacuated. You can also specify subdisks, columns, logs, mirrors or entire volumes to be evacuated. In these cases, you can use the `evac_storage` attribute to specify which storage is to be evacuated. In the following example, all columns of the volume are evacuated that have disks on controller `c1`:

```
# vxassist -g mydg evacuate column vol01 \
  evac_storage='"Controller"="c1"'
```

The next example evacuates any disks in columns 0 or 1 that lie on controller `c2`:

```
# vxassist -g mydg evacuate column vol01 column=0,1 \
  evac_storage='"Controller"="c1"'
```

This command specifies that both columns 0 and 1 are to be evacuated to disks on controller c2:

```
# vxassist -g mydg evacuate column vol01 column=0,1 \
  use_storage='"Controller"="c2"'
```

In the final example, volume data is evacuated from subdisks that are connected to controller `c1` to disks on any other controller:

```
# vxassist -g mydg evacuate subdisk vol01 \
  evac_storage='allof("Controller"="c1")' \
  use_storage='noneof("Controller"="c1")'
```

If the specified volume is currently enabled, the data in enabled plexes and their component enabled plexes is moved without interrupting the availability of the volume and without changing its redundancy. Subdisks that are within detached plexes, disabled plexes, detached logs, or RAID-5 subdisks are moved without any attempt to recover the data.

If the specified volume is not currently enabled, stale or offline plexes are moved without recovery. The evacuation fails if a non-enabled volume contains other subdisks that need to be moved.

Removing a volume

Once a volume is no longer required, you can use the following command to delete it and make its storage available for re-use:

```
# vxassist [-g diskgroup] remove volume volume
```

For example, the following command removes the volume, `vol1`:

```
# vxassist -g mydg remove volume vol1
```

Performing online relayout on a volume

ISP does not support online relayout of an application volume in the same way as when the `vxassist relayout` command is used on a traditional volume. ISP can perform relayout internally if this is necessary to preserve the intent of a volume, or to support operations such as changing the number of mirrors, columns or logs that are associated with a volume. However, you should note that such operations may destroy the intent of a volume by changing its desired data redundancy or performance capabilities.

The nearest equivalent in ISP to the online relayout operation is capability transformation. This changes the capabilities of a volume in a controlled fashion, and preserves the size of the volume but not its intent. This operation is

discussed in the following section. Subsequent sections describe how to add or remove mirrors, columns or logs to or from a volume, how to stop and reverse transformation and relayout operations, and how to verify that the intent of a volume has been preserved.

Transforming the capabilities of a volume online

To change the capabilities of an existing volume online, use the `vxassist transform` command as shown here:

```
# vxassist [-g diskgroup] [-b] transform volume [attributes...]
```

If specified, the `-b` option performs the transformation in the background.

For example, the following command changes the capability of the volume, `vol02`, to DataMirroring with 4 mirrors:

```
# vxassist -g mydg -b transform vol02 \
  capability='DataMirroring (nmirs=4)'
```

The next example adds the InstantSnapShottable and DCOLogMirroring capabilities to enable the use of instant snapshots with a mirrored volume:

```
# vxassist -g mydg -b transform mir3vol \
  capability='DataMirroring(nmirs=3),InstantSnapShottable,\
DCOLogMirroring(nlogs=3)'
```

Adding mirrors to a volume

To add mirrors to an existing volume, use the following command:

```
# vxassist [-g diskgroup] [-b] mirror volume \
  [nmirrors=number] [attributes...]
```

By default, one mirror is added. You can use the `nmirrors` attribute to specify the number of mirrors to add. If specified, the `-b` option performs the synchronization of the new mirrors in the background. You can also use storage attributes to specify the storage to be used for the mirrors. For example, the following command adds a mirror to the volume, `mirvol1`, using disk `mydg01`:

```
# vxassist -g mydg -b mirror mirvol1 \
  use_storage='"DM"="mydg01"'
```

Removing mirrors from a volume

To remove mirrors from a volume, use the following command:

```
# vxassist [-g diskgroup] [-f] remove mirror volume \
  [nmirrors=number] [attributes...]
```

By default, one mirror is removed. You can use the `nmirrors` attribute to specify the number of mirrors to remove. You can use storage attributes to specify the

storage to be removed. For example, the following example removes the mirror on the disk `mydg01`, from the volume `mirvol1`:

```
# vxassist -g mydg remove mirror mirvol1 \
  remove_storage='"DM"="mydg01"'
```

Note: If you use the `vxassist` command to remove mirrors, you must specify the `-f` (force) option to the command if the operation would violate any rules. For example, the rules may imply that a volume must have minimum number of mirrors of a particular kind.

Adding columns to a volume

To add columns to a striped or RAID-5 volume, use the following command:

```
# vxassist [-g diskgroup] [-b] add column volume ncols=number \
  [tmplen=length] [attributes...]
```

The `ncols` attribute specifies the number of columns to add. You can use storage attributes to specify the storage to be used for the columns. If specified, the `-b` option adds the columns in the background.

Note: This operation creates a temporary 2-way mirror volume that requires at least two disks, and therefore twice as much disk space as the size of the temporary volume. The size of the temporary volume is based on the size of the volume that is being changed. If required, you can use the `tmplen` attribute to specify the size of the temporary volume.

For example, the following command adds a column to the volume, `strpvol1`, using disk `enc1_5`:

```
# vxassist -g mydg -b add column strpvol1 ncols=1 \
  use_storage='"DeviceName"="enc1_5"'
```

Note: If an HFS or Base JFS (Lite VxFS) file system is configured on an ISP volume, do not specify `layout=grow` to the `vxvoladm` command when adding columns to the volume unless you first unmount the file system. These file system types can only be grown if they are first unmounted. This restriction does not apply to an Online JFS (Full VxFS) file system, which can be grown while mounted. See the `vxresize`(1M) manual page for more information.

Removing columns from a volume

To remove columns from a striped or RAID-5 volume, use the following command:

```
# vxassist [-g diskgroup] [-b] [-f] remove column volume \
    [ncols=number] [tmplen=length]
```

The `ncols` attribute specifies the number of columns to remove. If specified, the `-b` option removes the columns in the background.

Note: This operation creates a temporary 2-way mirror volume that requires at least two disks, and therefore twice as much disk space as the size of the temporary volume. The size of the temporary volume is based on the size of the volume that is being changed. If required, you can use the `tmplen` attribute to specify the size of the temporary volume.

For example, the following command removes a column from the volume, `strpvol1`:

```
# vxassist -g mydg -b remove column strpvol1 ncols=1
```

Note: If you use the `vxvoladm` command to remove columns, you must specify the `-f` (force) option to the command if the operation would violate any rules. For example, the rules may imply that a volume must have minimum number of columns of a particular kind.

If an HFS or Base JFS (Lite VxFS) file system is configured on an ISP volume, do not specify `layout=shrink` to the `vxvoladm` command when removing columns from the volume. These file system types cannot be shrunk. This restriction does not apply to an Online JFS (Full VxFS) file system, which can be shrunk while mounted. See the `vxresize`(1M) manual page for more information.

Changing the stripe unit size of volumes

To change the stripe unit size of a striped or RAID-5 volume, use the following command:

```
# vxassist [-g diskgroup] [-b] setstwidth volume \
  stripeunit=size [tmplen=length] [attributes...]
```

If specified, the `-b` option changes the stripe width in the background.

Note: This operation creates a temporary 2-way mirror volume that requires at least two disks, and therefore twice as much disk space as the size of the temporary volume. The size of the temporary volume is based on the size of the volume that is being changed. If required, you can use the `tmplen` attribute to specify the size of the temporary volume.

For example, the following command changes the stripe unit size of the volume `vol1` to 32KB:

```
# vxassist -g mydg -b setstwidth vol1 stripeunit=32k
```

Adding logs to a volume

To add logs to a volume, use the following command:

```
# vxassist [-g diskgroup] [-b] addlog volume logtype=type \
  [nlogs=number] [loglen=length] [attributes...]
```

The supported log types are `dco`, `drl` and `raid5`.

Adding a data change object (DCO) and DCO plex to a volume also allows the use of instant snapshots and dirty region logging (DRL) with the volume.

Traditional DRL may be added by specifying `logtype=drl` instead of `logtype=dco`. Such logs are implemented using a dedicated plex for each log copy, rather than using a DCO volume, and are supported for mirrored volume layouts only.

RAID-5 logs are supported for software RAID-5 volumes only.

By default, one log or a DCO with one plex is added. You can use the `nlogs` attribute to specify the number of logs or DCO plexes to add. The `loglen` attribute can be used to specify the size of a RAID-5 log or a DCO volume. If the volume already contains such a log or DCO volume, this attribute is ignored. For a DCO volume, ISP may round up the length to make room for the maps that are required. For a RAID-5 log, a minimum size of three times the full stripe width is imposed, and the length is rounded up to a integral multiple of the full stripe width.

You can use storage attributes to specify the storage to be used for the logs. For example, the following command adds a DCO plex to the volume, `mirvol1`, using disk `enc1_7`:

```
# vxassist -g mydg -b addlog mirvol1 logtype=dco \
  use_storage='"DeviceName"="enc1_7"'
```

Removing logs from a volume

To remove logs from a volume, use the following command:

```
# vxassist [-g diskgroup] [-f] remove log volume \
  [nlogs=number] logtype=type [attributes...]
```

The supported log types are `dco` and `raid5`. By default, one RAID-5 log or one plex of a DCO volume is removed. You can use the `nlogs` attribute to specify the number of logs or DCO plexes to remove. You can use storage attributes to specify the storage to be removed. In the following example, a DCO plex on the disk `mydg11` is removed from the volume `mirvol1`:

```
# vxassist -g mydg remove log mirvol1 logtype=dco\
  remove_storage='"DM"="mydg11"'
```

To remove all logs of a particular type from a volume, use the following command:

```
# vxassist [-g diskgroup] removeall log volume logtype=type
```

Note: If you use the `vxassist` command to remove logs, you must specify the `-f` (force) option to the command if the operation would violate any rules. For example, the rules may imply that a volume must have minimum number of logs of a particular kind. You must also specify the `-f` option when removing a DCO that is in use by DRL configured on a volume.

Monitoring and controlling ISP tasks

ISP performs management of objects (such as subdisks, plexes, and volumes). Once these objects have been created, VxVM can start performing I/O with them.

Note: The online transformation of an ISP volume is not necessarily complete if the `vxtask` command shows that synchronization of the volume has finished. A small additional time is required to perform cleanup operations.

For example, if you create a 2-way mirrored volume in the background, ISP creates an allocation task. When ISP has allocated storage for the volume, it lays out the volume on that storage and then starts the volume. At this point,

VxVM takes over control of I/O, and it begins to initialize the volume by synchronizing its plexes. To view the progress of this synchronization, you would use the `vxtask monitor` command.

See the "Monitoring and Controlling Tasks" section in the "Administering Volumes" chapter of the *Veritas Volume Manager Administrator's Guide*.

See the `vxtask`(1M) manual page.

Reversing volume transformations

If a volume transformation, which was invoked using the `vxassist transform` command, is in progress, you can use the `vxassist transformreverse` command to stop and reverse the transformation:

```
# vxassist [-g diskgroup] transformreverse volume
```

This command can also be used to reverse the following operations on mirrored-stripe volumes:

- Adding a column to a mirrored-stripe volume (`vxassist add column`).
- Removing a column from a mirrored-stripe volume (`vxassist remove column`).
- Changing the stripe width of a mirrored-stripe volume (`vxassist setstwidth`).

To stop and reverse the relayout, add column, remove column and setstwidth operations on striped, striped-mirror and RAID-5 volumes

1. Enter the following command to discover the task tag of the operation that you want to reverse:

   ```
   # vxtask list
   ```

2. Use the task tag with this form of the `vxtask` command to stop the operation:

   ```
   # vxtask abort task_tag
   ```

3. Finally, use the `vxrelayout` command to revert the volume to its former layout:

   ```
   # vxrelayout [-g diskgroup] reverse volume
   ```

Note: When mirrors are added or removed, ISP does not use the `relayout` or `transform` operations internally. After adding a mirror to a volume, VxVM starts synchronizing the new plexes from the existing plexes. In this case, the reverse operation can be performed by removing the newly added plexes. To reverse the removal of a mirror, a new plex must be added, and time allowed for it to be brought into synchronization with the volume.

Finding volumes that use specified capabilities or templates

To list the volumes that were created using certain capabilities or templates, use the following command:

```
# vxassist [-g diskgroup] [-P storage_pool] list \
  [capability=c1[,c2...]] [template=t1[,t2...]]
```

For example, to list all the volumes in the disk group `mydg` that implement the `DataMirroring` capability:

```
# vxassist -g mydg list capability=DataMirroring
TY NAME     DG      POOL    LENGTH  LAYOUT          NMIR    NCOL
v  vol1     mydg    data    10g     stripe-mirror   2       10
v  vol2     mydg    data    500m    mirror          2       -
```

The length of the volume is rounded down to the nearest whole number of standard units (`k` for kilobytes, `m` for megabytes, `g` for gigabytes, and so on).

Verifying the intent of a volume

To verify whether the application volumes in a disk group conform to the template rules, user-defined rules, volume group rules and storage pool rules that were used to create them, use the following command:

```
# vxassist -g diskgroup verify
```

To verify the intent of all application volumes in all disk groups, use this form of the command:

```
# vxassist -a verify
```

Displaying the rules associated with volumes

You can use the following command to display the capabilities, volume templates and rules that were specified when a volume was created:

```
# vxassist [-g diskgroup] printintent volume ...
```

To display the complete set of rules that were used to create a volume, use the following command:

```
# vxassist [-g diskgroup] printrules volume ...
```

If a rule is marked as `desired`, the output indicates whether the rule was obeyed or not.

The `vxassist printintent` command displays the capabilities, volume templates and rules that were specified when a volume was created. The `vxassist printrules` command displays the final set of rules that were obtained by precessing the specified capabilities, volume templates and rules. This final set of rules is what is used to allocate storage to a volume when it is created.

Note: New volumes that are created using the output from running the `vxassist printrules` command on an existing volume are allocated storage in exactly the same way as for that volume. If you use the output from the `vxassist printintent` command to create new volumes, the rule set is regenerated from the capabilities and templates. This may give a different result while still preserving the intent of the volume as advertised by the capabilities and templates.

Migrating non-ISP volumes to ISP volumes

To make non-ISP volumes that you previously created into volumes that are manageable by ISP, you can use the following form of the `vxassist` command:

```
# vxassist [-g diskgroup] [-P storage_pool] [-I rules] \
  migrate volume ...
```

Note: Volumes to be migrated must be CDS-compliant and have an alignment value of 8k.

The volumes to be migrated from a disk group may be specified on the command line, or in a definitions file that is read from the standard input if the `-M` option is specified:

```
# vxassist [-g diskgroup] -M migrate
```

The syntax of the definitions file is shown below:

```
diskgroup "dgname"
volume "volumename1" {
    pool "pool_name1"
    rules { volume1_rules }
};
volume "volumename2" {
    pool "pool_name2"
rules { volume2_rules }
};
.
.
.
```

The default intent that ISP applies to the volumes that are being migrated is to allocate plexes, logs and columns on separate disks. You can specify additional rules using the `-I` option on the command line, or by entries in the definitions file. You cannot include mirror, stripe or log rules. After the volumes have been

migrated, their intent is preserved by ISP as for volumes that are created directly using the `vxassist` or `vxvoladm` commands.

Rules specified on the command line are applied to all the specified volumes. Using a definitions file allows you to apply different rules to each volume.

Volumes that are specified on the command line are also associated with the same storage pool. If a storage pool is not specified using the `-P` option, the volumes are placed in the data storage pool of the disk group. Using a definitions file allows the migrated volumes to be placed in different storage pools.

The following example demonstrates how to use a definitions file to migrate two volumes to separate data and clone storage pools:

```
vxassist -M migrate <<!!
diskgroup "mydg"
volume "myvol" {
    pool "mydatapool"
};
volume "snap_myvol" {
    pool "myclonepool"
};
!!
```

Note the following limitations of the migration operation:

- All non-ISP volumes that share a set of disks must be migrated in a single operation.
- Disks cannot be shared with any volumes that are not listed on the command line or in a definitions file.
- Volumes cannot share disks unless these disks are to be located in the same storage pool.
- Volumes with a usage type of root, such as those on a root disk that is under VxVM control, cannot be migrated.
- The disk group containing the volumes to be migrated must have a version number of 120 or greater.

Migrating ISP volumes to non-ISP volumes

The `vxassist reversemigrate` operation allows you to convert ISP volumes to non-ISP volumes. The converted volumes permanently lose any intent that was associated with them when they were ISP volumes, and the `vxassist` command cannot be used to administer the configuration of any storage that is associated with them.

Use the following command to migrate ISP volumes to non-ISP volumes:

```
# vxassist [-g diskgroup] reversemigrate volume ...
```

Note: The volumes that are specified on the command line must not share any disks with any volumes that are not listed on the command ine.

To migrate all ISP volumes in a disk group to non-ISP volumes, use this form of the command:

```
# vxassist [-g diskgroup] reversemigrateall
```

Chapter 5

Administering instant snapshots

Veritas Volume Manager (VxVM) provides the capability for taking an image of a volume at a given point in time. Such an image is referred to as a *volume snapshot*.

Instant volume snapshots allow you to make backup copies of your volumes online with minimal interruption to users. You can then use the backup copies to restore data that has been lost due to disk failure, software errors or human mistakes, or to create replica volumes for the purposes of report generation, application development, or testing.

Three kinds of instant volume snapshots are supported for use with application volumes created by Veritas Intelligent Storage Provisioning (ISP):

- Full-sized instant snapshots require an empty volume to be prepared for use as the snapshot volume. The length of this volume must be the same as that of the volume whose snapshot is being taken. A snapshot volume can be created in either a data pool or in a clone pool.
- Linked break-off snapshot volumes differ from full-sized instant snapshots in that they can be set up in a different disk group from the data volume. This makes them especially suitable for off-host processing applications where you may want to create the snapshot on storage with different characteristics from that used for the data volumes.
- Space-optimized instant snapshots require less space than a full-sized instant snapshot but they cannot be dissociated from their original volume nor can they be moved into a different disk group. A space-optimized snapshot uses a storage pool or cache, which can be shared by one or more other space-optimized snapshots. You must set up the cache and the space-optimized snapshot volume in advance of creating any space-optimized snapshots.

See “Creating instant snapshots” on page 101.

For details of how to use volume snapshots to implement off-host online backup, see the *Veritas Storage Foundation Intelligent Storage Provisioning Solutions Guide*.

For more information about instant volume snapshot features, see the chapter “Understanding Veritas Volume Manager” in the *Veritas Volume Manager Administrator’s Guide*.

Full details of how to recover from failures of instant snapshot commands may be found in the “Recovery from failure of instant snapshot operations’’ chapter of the *Veritas Volume Manager Troubleshooting Guide*.

Note: Most VxVM commands require superuser or equivalent privileges.

Limitations of volume snapshots

A volume snapshot represents the data that exists in a volume at a given point in time. As such, VxVM does not have any knowledge of data that is cached by the overlying file system, or by applications such as databases that have files open in the file system. If the `fsgen` volume usage type is set on a volume that contains a Veritas File System (VxFS), intent logging of the file system metadata ensures the internal consistency of the file system that is backed up. For other file system types, depending on the intent logging capabilities of the file system, there may be inconsistencies between in-memory data and the data in the snapshot image.

For databases, a suitable mechanism must additionally be used to ensure the integrity of tablespace data when the volume snapshot is taken. The facility to temporarily suspend file system I/O is provided by most modern database software. For ordinary files in a file system, which may be open to a wide variety of different applications, there may be no way to ensure the complete integrity of the file data other than by shutting down the applications and temporarily unmounting the file system. In many cases, it may only be important to ensure the integrity of file data that is not in active use at the time that you take the snapshot.

Traditional third-mirror break-off snapshots cannot be taken of application volumes that have been created by ISP.

Preparing storage pools for full-sized instant snapshots

Assuming that you have already set up a data storage pool in a disk group, you can create one or more clone storage pools in the same disk group by using the same procedure. You may choose to use different types of disks or templates with a clone storage pool than you set up for the data storage pool.

For example, the following commands set up a data and clone storage pool in the disk group, `ttdg`, from storage pool definitions:

```
# vxpool -g ttdg create ttdtpool \
  pooldefinition=prefab_mirrored_volumes

# vxpool -g ttdg create ttclpool \
  pooldefinition=mirrored_volumes
```

The data storage pool, `ttdtpool`, is configured with templates that support the creation of volumes on mirrored disks that are prefabricated in hardware. The templates that are installed for the clone storage pool, `ttclpool`, support mirrored volumes that may be created using software. Having set up the storage pools, they can be populated with initialized disks.

See “Adding disks to a storage pool” on page 55.

Several storage pool sets are provided for setting up data and clone storage pools within a disk group. To set up a data and clone storage pool using one of these defined sets, use the following command:

```
# vxpool [-g diskgroup] organize storage_pool_set
```

For example, to set up a data storage pool that supports mirrored volumes and a clone storage pool that supports striped snapshots in the disk group, `mydg`, you could use this command:

```
# vxpool -g mydg organize mirrored_data_striped_clones
```

This command assigns default names for the storage pools. These names are based on the definition names. If required you can rename these storage pools.

See “Storage pool set” on page 195.

See “Renaming a storage pool” on page 58.

Creating a volume for use as a full-sized instant or linked break-off snapshot

If you want to create a full-sized instant or linked break-off snapshot for an original volume, you can use an empty volume with the required capability, and with the same length and region size as the original volume.

To create an empty volume for use by a full-sized instant or linked break-off snapshot

1 Use the `vxprint` command on the original volume to find the required size for the snapshot volume.

```
# LEN=`vxprint [-g diskgroup] -F%len volume`
```

Note: The command shown in this and subsequent steps assumes that you are using a Bourne-type shell such as `sh`, `ksh` or `bash`. You may need to modify the command for other shells such as `csh` or `tcsh`.

---

2 Use the `vxassist` command to create a volume, *snapvol*, of the required size and redundancy:

```
# vxassist [-g diskgroup] [-P storage_pool] make snapvol \
  $LEN [storage_specification ...] [attribute ...] \
  type=snapshot [regionsize=size] init=active
```

The attribute `regionsize` specifies the minimum size of each chunk (or region) of a volume whose contents are tracked for changes. The region size must be a power of 2, and be greater than or equal to 16KB. A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization. The default region size is 64k (64KB).

Note: If the region size of a space-optimized snapshot differs from the region size of the cache, this can degrade the system's performance compared to the case where the region sizes are the same.

The `init=active` attribute is specified to make the volume available immediately.

The following example creates a 10-gigabyte mirrored volume, `ttsnpvol`, in the clone storage pool, `ttclpool`:

```
# vxassist -g ttdg -P ttclpool make ttsnpvol 10g \
  capability='DataMirroring(nmirs=2)' type=snapshot \
  init=active
```

Creating a shared cache volume and preparing space-optimized snapshots

Note: If you intend to create space-optimized instant snapshots that share a cache volume, the region size that you specify for the volume must be greater than or equal to any region size that you specify for the cache volume. Creation of space-optimized snapshots that use a shared cache fails if the region size of the volume is smaller than the region size of the cache.

If the region size of a space-optimized snapshot differs from the region size of the cache, this can degrade the system's performance compared to the case where the region sizes are the same.

If you need to create several space-optimized instant snapshots for the volumes in a disk group, you may find it more convenient to create a single shared cache volume in the disk group rather than a separate cache volume for each snapshot.

To create a shared cache volume and prepare any space-optimized snapshots

1. Decide on the following characteristics that you want to allocate to the cache volume:
   - The size of the cache volume should be sufficient to record changes to the parent volumes during the interval between snapshot refreshes. A suggested value is 10% of the total size of the parent volumes for a refresh interval of 24 hours.
   - If redundancy is a desired characteristic of the cache volume, it should be mirrored. This increases the space required by the cache volume in proportion to the number of mirrors that it has.
   - If the cache volume is mirrored, space is required on at least as many disks as it has mirrors. These disks should not be shared with the disks used for the parent volumes. The disks should also be chosen to avoid impacting I/O performance for critical volumes, or hindering disk group split and join operations.
2. Having decided on its characteristics, use the `vxassist` command to create the cache volume:

   ```
   # vxassist [-g diskgroup] [-P storage_pool] make cachevol \
     size [storage_specification ...] [attribute ...] \
     [regionsize size] type=cachevolume
   ```

   The attribute `regionsize` specifies the minimum size of each chunk (or region) of a volume whose contents are tracked for changes. The region size

must be a power of 2, and be greater than or equal to 16KB. A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization. The default region size is 64k (64KB).

The following example creates a 1GB mirrored cache volume, `cachevol`, in the clone storage pool, `myclpool`, within the disk group, `mydg`:

```
# vxassist -g mydg -P myclpool make mycache 1g \
  rules="mirror 2" type=cachevolume
```

3 Once the cache volume has been created, use the following command to prepare each space-optimized snapshot that uses the cache volume:

```
# vxassist [-g diskgroup] [-P storage_pool] make sovol \
  srcvol_len [storage_specification ...] [attribute ...] \
  type=snapshot cachevolume=cachevol init=active
```

Note: The *srcvol_len* argument specifies the length of the source volume for which the snapshot is being prepared. This value defines the logical size of the snapshot. The actual amount of storage that the snapshot requires is less than this, and is limited by the size of the cache volume.

For example, this command creates a space-optimized snapshot volume that uses the cache volume, `mycache`:

```
# vxassist -g mydg -P myclpool make mysovol 10g \
  type=snapshot cachevolume=mycache
```

Note: The argument `10g` is the size of the source volume for which the snapshot `mysovol` is being prepared.

Once created, such a volume is ready for use to take a space-optimized instant snapshot.

Alternatively, you can use the following command to create a cache volume and prepare the space-optimized snapshot volumes in a single operation:

```
# vxassist -M make <<!!
    volumegroup {
            diskgroup "diskgroup"
            volume "cachevol" cache_size {
                    type cachevolume
                    [storage_specification]
                    [regionsize size]
            }
            volume "sovol1" vol1_size {
                    init active
                    cachevolume "cachevol"
            }
            volume "sovol2" vol2_size {
                    init active
                    cachevolume "cachevol"
            }
.
```

```
.
.
};
!!
```

Note: Because of its complexity, it is recommended that you run this command as a script.

For the examples given earlier in this section, the combined form of the command would be:

```
# vxassist -M make <<!!
    volumegroup {
            diskgroup "mydg"
            volume "mycache" 1g {
                    type cachevolume
                    capability 'DataMirroring(nmirs=2)'
            }
            volume "mysovol" 10g {
                    init active
                    cachevolume "cachevol"
            }
};
!!
```

Tuning the autogrow attributes

The `highwatermark`, `autogrowby` and `maxautogrow` attributes determine how the VxVM cache daemon (`vxcached`) maintains the cache if the `autogrow` feature has been enabled and `vxcached` is running:

- When cache usage reaches the high watermark value, `highwatermark` (default value is 90 percent), `vxcached` grows the size of the cache volume by the value of `autogrowby` (default value is 20% of the size of the cache volume in blocks). The new required cache size cannot exceed the value of `maxautogrow` (default value is twice the size of the cache volume in blocks).
- When cache usage reaches the high watermark value, and the new required cache size would exceed the value of `maxautogrow`, `vxcached` deletes the oldest snapshot in the cache. If there are several snapshots with the same age, the largest of these is deleted.

If the `autogrow` feature has been disabled:

- When cache usage reaches the high watermark value, `vxcached` deletes the oldest snapshot in the cache. If there are several snapshots with the same age, the largest of these is deleted. If there is only a single snapshot, this snapshot is detached and marked as invalid.

Note: The `vxcached` daemon does not remove snapshots that are currently open, and it does not remove the last or only snapshot in the cache.

If the cache space becomes exhausted, the snapshot is detached and marked as invalid. If this happens, the snapshot is unrecoverable and must be removed. Enabling the `autogrow` feature on the cache helps to avoid this situation occurring. However, for very small caches (of the order of a few megabytes), it is possible for the cache to become exhausted before the system has time to respond and grow the cache. In such cases, either increase the size of the cache manually, or use the `vxcache set` command to reduce the value of `highwatermark` for the cache object, as shown in this example:

```
# vxcache -g mydg set highwatermark=60 cobjmydg
```

See "Growing and shrinking a cache" on page 100.

You can use the `maxautogrow` attribute to limit the maximum size to which a cache can grow. To estimate this size, consider how much the contents of each source volume are likely to change between snapshot refreshes, and allow some additional space for contingency.

If necessary, you can use the `vxcache set` command to change other `autogrow` attribute values for a cache. See the `vxcache`(1M) manual page for details.

Note: Ensure that the cache is sufficiently large, and that the `autogrow` attributes are configured correctly for your needs.

Growing and shrinking a cache

You can use the `vxcache` command to increase the size of the cache volume that is associated with a cache object:

```
# vxcache [-g diskgroup] growcacheto cache_object size
```

For example, to increase the size of the cache volume associated with the cache, `mycache`, to 2GB, use the following command:

```
# vxcache -g mydg growcacheto mycache 2g
```

To grow a cache by a specified amount, use the following form of the command:

```
# vxcache [-g diskgroup] growcacheby cache_object size
```

For example, to increase the size of the cache, `mycache`, by 1GB, you would use the following command:

```
# vxcache -g mydg growcacheby mycache 1g
```

You can similarly use the `shrinkcacheby` and `shrinkcacheto` operations to reduce the size of a cache. See the `vxcache`(1M) manual page for more information.

Removing a cache

To remove a cache completely, including the cache object, its cache volume and all space-optimized snapshots that use the cache

1. Run the following command to find out the names of the top-level snapshot volumes that are configured on the cache object:

   ```
   # vxprint -g diskgroup -vne \
     "v_plex.pl_subdisk.sd_dm_name ~ /cache_object/"
   ```

   where *cache_object* is the name of the cache object.

2. Remove all the top-level snapshots and their dependent snapshots (this can be done with a single command):

   ```
   # vxedit -g diskgroup -r rm snapvol ...
   ```

   where *snapvol* is the name of a top-level snapshot volume.

3. Stop the cache object:

   ```
   # vxcache -g diskgroup stop cache_object
   ```

4. Finally, remove the cache object and its cache volume:

   ```
   # vxedit -g diskgroup -r rm cache_object
   ```

Creating instant snapshots

Note: Instant snapshots of ISP application volumes in a disk group's data pool are best created in clone pools that are also associated with the same disk group. A snapshot of a snapshot does not have to be in the same clone pool as its parent.

Volume sets can be used in place of volumes with the following `vxsnap` operations on instant snapshots: `addmir`, `dis`, `make`, `prepare`, `reattach`, `refresh`, `restore`, `rmmir`, `split`, `syncpause`, `syncresume`, `syncstart`, `syncstop`, `syncwait`, and `unprepare`. A snapshot of a volume set must itself be a volume set. A full-sized instant snapshot of a volume set can be created using a prepared volume set. You cannot use the `nmirrors` or `plex` attributes to specify that existing plexes are to be broken off. See the chapter "Creating and Administering Volume Sets" in the *Veritas Volume Manager Administrator's Guide* for more information on creating volume sets.

Note: You may need an additional license to use this feature.

VxVM allows you to make instant snapshots of volumes by using the `vxsnap` command.

A plex in a full-sized instant snapshot requires as much space as the original volume. If you instead make a space-optimized instant snapshot of a volume, this only requires enough storage to record the original contents of the parent volume as they are changed during the life of the snapshot.

The recommended approach to performing volume backup from the command line, or from a script, is to use the `vxsnap` command. The `vxsnap prepare` and `make` tasks allow you to back up volumes online with minimal disruption to users.

The `vxsnap prepare` step creates a DCO and DCO volume and associates this with the volume. It also enables Persistent FastResync on the volume.

The `vxsnap make` step creates an instant snapshot that is immediately available for making a backup. After the snapshot has been taken, read requests for data in the original volume are satisfied by reading either from a non-updated region of the original volume, or from the copy of the original contents of an updated region that have been recorded by the snapshot.

Note: Synchronization of a full-sized instant snapshot from the original volume is enabled by default. If you specify the `syncing=no` attribute to `vxsnap make`, this disables synchronization, and the contents of the instant snapshot are unlikely ever to become fully synchronized with the contents of the original volume at the point in time that the snapshot was taken. If you wish to move an instant snapshot volume to another disk group for export to another machine for off-host processing, or to turn it into an independent volume, you must ensure that the snapshot volume has been completely synchronized.

You can immediately retake a full-sized, space-optimized or linked break-off snapshot at any time by using the `vxsnap refresh` command. If a fully synchronized full-sized or linked break-off snapshot is required, you must wait for the new resynchronization to complete.

To create and manage a snapshot of a volume with the `vxsnap` command, follow the procedure in "Preparing to create instant and break-off snapshots" on page 103, and then use one of the procedures described in the following sections:

- "Creating and managing space-optimized instant snapshots" on page 104
- "Creating and managing full-sized instant snapshots" on page 106
- "Creating and managing linked break-off snapshot volumes" on page 108

Preparing to create instant and break-off snapshots

To prepare a volume for the creation of instant and break-off snapshots

1 Use the following commands to see if the volume is associated with a version 20 data change object (DCO) and DCO volume that allow instant snapshots and Persistent FastResync to be used with the volume, and to check that FastResync is enabled on the volume:

```
# vxprint -g volumedg -F%instant volume
# vxprint -g volumedg -F%fastresync volume
```

If both commands return a value of `on`, the volume can be used for instant snapshot operations, and you should skip to step 3. Otherwise continue with step 2.

2 To prepare a volume for instant snapshots, use the following command:

```
# vxsnap [-g diskgroup] prepare volume [regionsize=size] \
  [ndcomirs=number] [alloc=storage_attributes]
```

For example, to prepare the volume, `myvol`, in the disk group, `mydg`, use the following command:

```
# vxsnap -g mydg prepare myvol regionsize=128k ndcomirs=2 \
  alloc=mydg10,mydg11
```

This example creates a DCO object and redundant DCO volume with two plexes located on disks `mydg10` and `mydg11`, and associates them with `myvol`. The region size is also increased to 128KB from the default size of 64KB. The region size must be a power of 2, and be greater than or equal to 16KB. A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization.

3 If you intend to take a space-optimized instant snapshot of the volume, you must first set up a shared cache volume in the same disk group as the volume. This cache object for this cache volume can be maintained by using the `vxcache` command.

For full-sized instant snapshots, linked break-off snapshots and space-optimized instant snapshots, you must also create a volume for use as the snapshot volume. This volume must have the same region size as that of the volume for which the snapshot is being created. In addition, a volume that is created for use as a full-sized instant snapshot must be the same size as the volume for which the snapshot is being created.

See “Creating a shared cache volume and preparing space-optimized snapshots” on page 97.

See “Creating a volume for use as a full-sized instant or linked break-off snapshot” on page 96.

Note: If you intend to split the clone pool that contains snapshots into separate disk groups (for example, to perform off-host processing), the clone pool must only contain fully synchronized full-sized instant snapshots (which do not require a cache volume). You cannot split off a clone pool that contains either a cache volume or space-optimized instant snapshots.

Creating and managing space-optimized instant snapshots

Space-optimized instant snapshots are not suitable for write-intensive volumes (such as for database redo logs) because the copy-on-write mechanism may degrade the performance of the volume.

If you intend to split the volume and snapshot into separate disk groups (for example, to perform off-host processing), you must use a fully synchronized linked break-off snapshot (which does not require a cache object). You cannot use a space-optimized instant snapshot for this purpose.

Creation of space-optimized snapshots that use a shared cache fails if the region size specified for the volume is smaller than the region size set on the cache.

If the region size of a space-optimized snapshot differs from the region size of the cache, this can degrade the system's performance compared to the case where the region sizes are the same.

Note: For space-optimized instant snapshots that share a cache object, the specified region size must be greater than or equal to the region size specified for the cache object.

See "Creating a shared cache volume and preparing space-optimized snapshots" on page 97.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples; one for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (/). Tuples are separated by white space.

To create and manage a space-optimized instant snapshot

1 To create a space-optimized instant snapshot, *snapvol*, use the following forms of the `vxsnap make` command:

```
# vxsnap [-g diskgroup] make source=vol/snapvol=snapvol
```

For example, to create the space-optimized instant snapshot of the volume, `myvol`, in the disk group, `mydg`, and the prepared snapshot, `snap3myvol`, enter the following command:

```
# vxsnap -g mydg make source=myvol/snapvol=snap3myvol
```

Note: This step requires that you have created a shared cache object, and prepared a snapshot that uses this cache.

See "Creating a shared cache volume and preparing space-optimized snapshots" on page 97.

---

2 Use `fsck` (or some utility appropriate for the application running on the volume) to clean the temporary volume's contents. For example, you can use this command:

```
# fsck -F vxfs /dev/vx/rdsk/diskgroup/snapshot
```

3 If you require a backup of the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

4 You now have the following choices of what to do with a space-optimized instant snapshot:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized.
  See "Refreshing an instant snapshot" on page 111.
- Restore the contents of the original volume from the snapshot volume. The space-optimized instant snapshot remains intact at the end of the operation.
  See "Restoring a volume from an instant snapshot" on page 113.

Creating and managing full-sized instant snapshots

Full-sized instant snapshots are not suitable for write-intensive volumes (such as for database redo logs) because the copy-on-write mechanism may degrade the performance of the volume.

If you intend to split the volume and snapshot into separate disk groups (for example, to perform off-host processing), you must use a fully synchronized linked break-off snapshot. You cannot use a full-sized instant snapshot for this purpose with ISP volumes.

Note: For full-sized instant snapshots, you must prepare a volume that is to be used as the snapshot volume. This must be the same size as the volume for which the snapshot is being created, and it must also have the same region size.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 96.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples; one for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (/). Tuples are separated by white space.

To create and manage a full-sized instant snapshot

1 To create a full-sized instant snapshot, *snapvol*, use the following forms of the `vxsnap make` command:

```
# vxsnap [-g diskgroup] make source=volume/snapvol=snapvol\
[/syncing=off]
```

Note: This step requires that you have prepared an existing volume, *snapvol*, that is to be used as the snapshot volume.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 96.

Background synchronization of the snapshot volume from its parent volume is enabled by default (equivalent to specifying the `syncing=on` attribute). If you do not want to move the snapshot into a separate disk group, or turn it into an independent volume, specify the `syncing=off` attribute to disable synchronization. This avoids unnecessary system overhead.

For example, to use the prepared volume, `snap1myvol`, as the snapshot for the volume, `myvol`, in the disk group, `mydg`, use the following command:

```
# vxsnap -g mydg make source=myvol/snapvol=snap1myvol
```

If you want to turn a snapshot into an independent volume, you must wait for its contents to be synchronized with those of its parent volume.

You can use the `vxsnap syncwait` command to wait for the synchronization of the snapshot volume to be completed, as shown here:

```
# vxsnap [-g diskgroup] syncwait snapvol
```

For example, you would use the following command to wait for synchronization to finish on the snapshot volume, snap2myvol:

```
# vxsnap -g mydg syncwait snap2myvol
```

This command exits (with a return code of zero) when synchronization of the snapshot volume is complete. The snapshot volume may then be moved to another disk group or turned into an independent volume.

If required, you can use the following command to test if the synchronization of a volume is complete:

```
# vxprint [-g diskgroup] -F%incomplete snapvol
```

This command returns the value `off` if synchronization of the volume, *snapvol*, is complete; otherwise, it returns the value `on`.

See "Controlling instant snapshot synchronization" on page 118.

2 Use `fsck` (or some utility appropriate for the application running on the volume) to clean the temporary volume's contents. For example, you can use this command:

```
# fsck -F vxfs /dev/vx/rdsk/diskgroup/snapshot
```

3 If you require a backup of the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

You have the following choices for what to do with an instant snapshot:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized.

  See "Refreshing an instant snapshot" on page 111.

- Restore the contents of the original volume from the snapshot volume. For full instant snapshot volumes, you can choose whether none, a subset, or all of the plexes of the snapshot volume are returned to the original volume as a result of the operation. A space-optimized instant snapshot always remains intact at the end of the operation.

  See "Restoring a volume from an instant snapshot" on page 113.

- Dissociate the snapshot volume entirely from the original volume. This may be useful if you want to use the copy for other purposes such as testing or report generation. If desired, you can delete the dissociated volume.

  See "Dissociating an instant snapshot" on page 114 for details.

- If the snapshot is part of a snapshot hierarchy, you can also choose to split this hierarchy from its parent volumes.
  See "Splitting an instant snapshot hierarchy" on page 115.

Creating and managing linked break-off snapshot volumes

Linked break-off snapshots are the only form of volume snapshot that can be used to implement offhost processing for ISP volumes as described in the *Veritas Storage Foundation Intelligent Storage Provisioning Solutions Guide*. They are also the most suitable type of snapshot to use with write-intensive volumes, such as database redo logs.

For linked break-off snapshots, you must prepare a volume that is to be used as the snapshot volume. This must be the same size as the volume for which the snapshot is being created, and it must also have the same region size.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 96.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples; one for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (/). Tuples are separated by white space.

To create and manage a linked break-off snapshot

1 Use the following command to link the prepared snapshot volume, *snapvol*, to the data volume:

```
# vxsnap [-g diskgroup] [-b] addmir volume mirvol=snapvol \
  [mirdg=snapdg]
```

The optional `mirdg` attribute can be used to specify the snapshot volume's current disk group, *snapdg*. The `-b` option can be used to perform the synchronization in the background. If the `-b` option is not specified, the command does not return until the link becomes `ACTIVE`.

For example, the following command links the prepared volume, `prepsnap`, in the disk group, `mysnapdg`, to the volume, `vol1`, in the disk group, `mydg`:

```
# vxsnap -g mydg -b addmir vol1 mirvol=prepsnap \
  mirdg=mysnapdg
```

If the `-b` option is specified, you can use the `vxsnap snapwait` command to wait for the synchronization of the snapshot plexes or linked snapshot volume to complete, as shown in this example:

```
# vxsnap -g mydg snapwait vol1 mirvol=prepsnap \
  mirdg=mysnapvoldg
```

2 To create a linked break-off snapshot, use the following form of the `vxsnap make` command.

```
# vxsnap [-g diskgroup] make source=volume/snapvol=snapvol\
[/snapdg=snapdiskgroup]
```

The `snapdg` attribute must be used to specify the snapshot volume's disk group if this is different from that of the data volume.
For example, to use the prepared volume, `prepsnap`, as the snapshot for the volume, `vol1`, in the disk group, `mydg`, use the following command:

```
# vxsnap -g mydg make \
  source=vol1/snapvol=prepsnap/snapdg=mysnapdg
```

3 Use `fsck` (or some utility appropriate for the application running on the volume) to clean the temporary volume's contents. For example, you can use this command with a VxFS file system:

```
# fsck -F vxfs /dev/vx/rdsk/diskgroup/snapshot
```

4 If you require a backup of the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

5 You now have the following choices of what to do with a linked break-off snapshot:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized.
  See "Refreshing an instant snapshot" on page 111.

  Note: This operation is not possible if the linked volume and snapshot are in different disk groups.

- Relink the snapshot volume with the original volume.
  See "Reattaching a linked break-off snapshot volume" on page 112.
- Dissociate the snapshot volume entirely from the original volume. This may be useful if you want to use the copy for other purposes such as testing or report generation. If desired, you can delete the dissociated volume.
  See "Dissociating an instant snapshot" on page 114.
- If the snapshot is part of a snapshot hierarchy, you can also choose to split this hierarchy from its parent volumes.
  See "Splitting an instant snapshot hierarchy" on page 115.

Creating multiple instant snapshots

Note: In these examples, all snapshot volumes and any cache volumes must be prepared in advance.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 96.

See "Creating a shared cache volume and preparing space-optimized snapshots" on page 97.

To make it easier to create snapshots of several volumes at the same time, the `vxsnap make` command accepts multiple tuples that define the source and snapshot volumes names as their arguments. For example, to create three instant snapshots, each with the same redundancy, the following form of the command can be used:

```
# vxsnap [-g diskgroup] make \
  source=vol1/snapvol=snapvol1 \
  source=vol2/snapvol=snapvol2 \
  source=vol3/snapvol=snapvol3
```

The specified source volumes (*vol1*, *vol2* and so on) may be the same volume or different volumes.

The `vxsnap make` command allows the snapshots to be of different types as shown here:

```
# vxsnap [-g diskgroup] make \
  source=vol1/snapvol=snapvol1 \
  source=vol2/snapvol=snapvol2
```

In this example, `snapvol1` is a full-sized snapshot, and `snapvol2` is a space-optimized snapshot.

Removing a linked break-off snapshot volume

To remove a linked break-off snapshot volume from a volume, use this command:

```
# vxsnap [-g diskgroup] rmmir volume|volume_set mirvol=snapvol \
  [mirdg=snapdiskgroup]
```

The `mirvol` and optional `mirdg` attributes specify the snapshot volume, *snapvol*, and its disk group, *snapdiskgroup*. For example, the following command removes a linked snapshot volume, `prepsnap`, from the volume, `vol1`:

```
# vxsnap -g mydg rmmir vol1 mirvol=prepsnap mirdg=mysnapdg
```

Adding a snapshot to a cascaded snapshot hierarchy

To create a snapshot and push it onto a snapshot hierarchy between the original volume and an existing snapshot volume, specify the name of the existing snapshot volume as the value of the `infrontof` attribute to the `vxsnap make` command. The following example shows how to place the space-optimized snapshot, `thurs_bu`, of the volume, `dbvol`, in front of the earlier snapshot, `wed_bu`:

```
# vxsnap -g dbdg make source=dbvol/snapvol=thurs_bu/\
infrontof=wed_bu/cache=dbdgcache
```

Similarly, the next snapshot that is taken, `fri_bu`, is placed in front of `thurs_bu`:

```
# vxsnap -g dbdg make source=dbvol/snapvol=fri_bu/\
infrontof=thurs_bu/cache=dbdgcache
```

For more information on the application of cascaded snapshots, see the *Veritas Volume Manager Administrator's Guide*.

Refreshing an instant snapshot

Refreshing an instant snapshot replaces it with another point-in-time copy of a parent volume. To refresh one or more snapshots and make them immediately available for use, use the following command:

```
# vxsnap [-g diskgroup] refresh snapvol [source=vol] \
  [[snapvol2 source=vol2]...] [sync=yes|no]
```

If the source volume is not specified, the immediate parent of the snapshot is used. For full-sized instant snapshots, resynchronization is started by default. To disable resynchronization, specify the `syncing=no` attribute. This attribute is not supported for space-optimized snapshots.

Note: The snapshot being refreshed must not be open to any application. For example, any file system configured on the volume must first be unmounted.

It is possible to refresh a volume from an unrelated volume provided that their sizes are compatible.

You can use the `vxsnap syncwait` command to wait for the synchronization of the snapshot volume to be completed, as shown here:

```
# vxsnap [-g diskgroup] syncwait snapvol
```

See "Controlling instant snapshot synchronization" on page 118.

Attaching plexes of an instant snapshot

Note: Although this operation is named `reattach`, the plexes of an ISP snapshot volume could never have belonged to the parent volume. For this reason, the description below refers to the *attachment*, rather than the *reattachment*, of snapshot plexes.

This operation is not supported for space-optimized instant snapshots.

See "Reattaching a linked break-off snapshot volume" on page 112.

Using the following command, some or all plexes of an instant snapshot may be attached to the specified source volume, or to a source volume in the snapshot hierarchy above the snapshot volume:

```
# vxsnap [-g diskgroup] reattach snapvol source=vol \
  [nmirror=number]
```

By default, all the plexes are attached, which results in the removal of the snapshot. If required, the number of plexes to be attached may be specified as the value assigned to the `nmirror` attribute.

Note: The snapshot being reattached must not be open to any application. For example, any file system configured on the snapshot volume must first be unmounted.

Only the plexes of a snapshot volume that was set up using ISP may be attached.

If the snapshot volume and the source volume lie in different storage pools, the underlying disks of the snapshot volume are moved to the source volume's storage pool. However, if other objects in the snapshot volume's storage pool are also configured on these disks, the `reattach` command fails.

For example the following command attaches one plex from the snapshot volume, `snapmyvol`, to the volume, `myvol`:

```
# vxsnap -g mydg reattach snapmyvol source=myvol nmirror=1
```

Reattaching a linked break-off snapshot volume

Unlike other types of snapshot, the reattachment operation for linked break-off snapshot volumes does not return the plexes of the snapshot volume to the parent volume. The link relationship is re-established that makes the snapshot volume a mirror of the parent volume, and this allows the snapshot data to be resynchronized. However, the snapshot volume is only readopted by its parent volume if they are both in the same disk group.

To reattach a linked break-off snapshot volume, use the following form of the `vxsnap reattach` command:

```
# vxsnap [-g snapdiskgroup] reattach snapvolume|snapvolume_set \
  source=volume|volume_set [sourcedg=diskgroup]
```

The `sourcedg` attribute must be used to specify the data volume's disk group if this is different from the snapshot volume's disk group, *snapdiskgroup*.

Note: The snapshot being reattached must not be open to any application. For example, any file system configured on the snapshot volume must first be unmounted.

It is possible to reattach a volume to an unrelated volume provided that their sizes and region sizes are compatible.

For example the following command reattaches the snapshot volume, `prepsnap`, in the disk group, `snapdg`, to the volume, `myvol`, in the disk group, `mydg`:

```
# vxsnap -g snapdg reattach prepsnap source=myvol sourcedg=mydg
```

After resynchronization of the snapshot volume is complete, it is placed in the `ACTIVE` state. You can use the `vxsnap snapwait` command (but not `vxsnap syncwait`) to wait for the resynchronization of the reattached volume to complete, as shown here:

```
# vxsnap -g snapdg snapwait myvol mirvol=prepsnap
```

Restoring a volume from an instant snapshot

On occasion, it may be desirable to reinstate the contents of a volume from a backup or modified replica within a snapshot volume. The following command may be used to restore a volume:

```
# vxsnap [-g diskgroup] restore [nmirrors=number] vol \
  [source=snapvol] [destroy=yes|no]
```

For a full instant snapshot, some or all of its plexes may be reattached to the parent volume or to a specified source volume in the snapshot hierarchy above the snapshot volume. If `destroy=yes` is specified, all the plexes of the snapshot are reattached and the snapshot volume is removed.

For a space-optimized instant snapshot, the cached data is used to restore the contents of the specified volume. The snapshot itself remains unchanged by the operation.

Note: For this operation to succeed, the volume that is being restored and the snapshot volume must not be open to any application. For example, any file systems that are configured on either volume must first be unmounted.

It is not possible to restore a volume from an unrelated volume.

The `destroy` and `nmirror` attributes are not supported for space-optimized instant snapshots.

The following example demonstrates how to restore the volume, `myvol`, from the space-optimized snapshot, `snap3myvol`.

```
# vxsnap -g mydg restore myvol source=snap3myvol
```

Dissociating an instant snapshot

The following command breaks the association between a snapshot volume, *snapvol*, and its parent volume, so that the snapshot may be used as an independent volume:

```
# vxsnap [-f] [-g diskgroup] dis snapvol
```

This operation fails if the snapshot, *snapvol*, has a snapshot hierarchy below it that contains unsynchronized snapshots. If this happens, the dependent snapshots must be fully synchronized from *snapvol*. When no dependent snapshots remain, *snapvol* may be dissociated. The snapshot hierarchy is then adopted by *snapvol*'s parent volume.

Note: To be usable after dissociation, the snapshot volume and any snapshots in the hierarchy must have been fully synchronized.

See "Controlling instant snapshot synchronization" on page 118.

In addition, you cannot dissociate a snapshot if synchronization of any of the dependent snapshots in the hierarchy is incomplete. If an incomplete snapshot is dissociated, it is unusable and should be deleted.

See "Removing an instant snapshot" on page 115.

The following command dissociates the snapshot, `snap2myvol`, from its parent volume:

```
# vxsnap -g mydg dis snap2myvol
```

Note: When applied to a volume set or to a component volume of a volume set, this operation can result in inconsistencies in the snapshot hierarchy in the case of a system crash or hardware failure. If the operation is applied to a volume set, the `-f` (force) option must be specified.

Removing an instant snapshot

When you have dissociated a full-sized instant snapshot, you can use the `vxassist` command to delete it altogether, as shown in this example:

```
# vxassist -g mydg remove volume snap2myvol
```

You can also use this command to remove a space-optimized instant snapshot from its cache.

See "Removing a cache" on page 101.

Splitting an instant snapshot hierarchy

Note: This operation is not supported for space-optimized instant snapshots.

The following command breaks the association between a snapshot hierarchy that has the snapshot volume, *snapvol*, at its head, and its parent volume, so that the snapshot hierarchy may be used independently of the parent volume:

```
# vxsnap [-f] [-g diskgroup] split snapvol
```

Note: The topmost snapshot volume in the hierarchy must have been fully synchronized for this command to succeed. Snapshots that are lower down in the hierarchy need not have been fully resynchronized.

See "Controlling instant snapshot synchronization" on page 118.

The following command splits the snapshot hierarchy under `snap2myvol` from its parent volume:

```
# vxsnap -g mydg split snap2myvol
```

Note: When applied to a volume set or to a component volume of a volume set, this operation can result in inconsistencies in the snapshot hierarchy in the case of a system crash or hardware failure. If the operation is applied to a volume set, the `-f` (force) option must be specified.

Displaying instant snapshot information

The vxsnap print command may be used to display information about the snapshots that are associated with a volume.

```
# vxsnap [-g diskgroup] print [vol]
```

This command shows the percentage progress of the synchronization of a snapshot or volume. If no volume is specified, information about the snapshots for all the volumes in a disk group is displayed.

The following example shows a volume, vol1, which has a full-sized snapshot, snapvol1 whose contents have not been synchronized with vol1:

```
# vxsnap -g mydg print
   NAME    SNAPOBJECT    TYPE    PARENT   SNAPSHOT  %DIRTY  %VALID

   vol1    --            volume  --       --        --      100
           snapvol1_snp1 volume  --       snapvol1  1.30    --
   snapvol1vol1_snp1     volume  vol1         --    1.30    1.30
```

The %DIRTY value for snapvol1 shows that its contents have changed by 1.30% when compared with the contents of vol1. As snapvol1 has not been synchronized with vol1, the %VALID value is the same as the %DIRTY value. If the snapshot were partly synchronized, the %VALID value would lie between the %DIRTY value and 100%. If the snapshot were fully synchronized, the %VALID value would be 100%. The snapshot could then be made independent or moved into another disk group.

Additional information about the snapshots of volumes and volume sets can be obtained by using the -n option with the vxsnap print command:

```
# vxsnap [-g diskgroup] -n [-l] [-v] [-x] print [vol]
```

Alternatively, you can use the vxsnap list command, which is an alias for the vxsnap -n print command:

```
# vxsnap [-g diskgroup] [-l] [-v] [-x] list [vol]
```

The following output is an example of using this command on the disk group dg1:

```
# vxsnap -g dg -vx list

NAME    DG    OBJTYPE SNAPTYPE  PARENT  PARENTDG  SNAPDATE        CHANGE_DATA  SYNCED_DATA
vol     dg1   vol     -         -       -         -               -            10G (100%)
svol1   dg2   vol     fullinst  vol     dg1       2006/2/1 12:29 20M (0.2%)   60M (0.6%)
svol2   dg1   vol     mirbrk    vol     dg1       2006/2/1 12:29 120M (1.2%)  10G (100%)
svol3   dg2   vol     volbrk    vol     dg1       2006/2/1 12:29 105M (1.1%)  10G (100%)
svol21  dg1   vol     spaceopt  svol2   dg1       2006/2/1 12:29 52M (0.5%)   52M (0.5%)
vol-02  dg1   plex    snapmir   vol     dg1       -               -            56M (0.6%)
mvol    dg2   vol     mirvol    vol     dg1       -               -            58M (0.6%)
vset1   dg1   vset    -         -       -         -               -            2G (100%)
v1      dg1   compvol -         -       -         -               -            1G (100%)
v2      dg1   compvol -         -       -         -               -            1G (100%)
svset1  dg1   vset    mirbrk    vset    dg1       2006/2/1 12:29 1G (50%)     2G (100%)
sv1     dg1   compvol mirbrk    v1      dg1       2006/2/1 12:29 512M (50%)   1G (100%)
sv2     dg1   compvol mirbrk    v2      dg1       2006/2/1 12:29 512M (50%)   1G (100%)
vol-03  dg1   plex    detmir    vol     dg1       -               20M (0.2%)   -
mvol2   dg2   vol     detvol    vol     dg1       -               20M (0.2%)   -
```

This shows that the volume vol has three full-sized snapshots, svol1, svol2 and svol3, which are of types full-sized instant (fullinst), mirror break-off (mirbrk) and linked break-off (volbrk). It also has one snapshot plex (snapmir), vol-02, and one linked mirror volume (mirvol), mvol. The snapshot svol2 itself has a space-optimized instant snapshot (spaceopt), svol21. There is also a volume set, vset1, with component volumes v1 and v2. This volume set has a mirror break-off snapshot, svset1, with component volumes sv1 and sv2. The last two entries show a detached plex, vol-03, and a detached mirror volume, mvol2, which have vol as their parent volume. These snapshot objects may have become detached due to an I/O error, or, in the case of the plex, by running the vxplex det command.

The CHANGE_DATA column shows the approximate difference between the current contents of the snapshot and its parent volume. This corresponds to the amount of data that would have to be resynchronized to make the contents the same again.

The SYNCED_DATA column shows the approximate progress of synchronization since the snapshot was taken.

The -l option can be used to obtain a longer form of the output listing instead of the tabular form.

The -x option expands the output to include the component volumes of volume sets.

See the vxsnap(1M) manual page for more information about using the vxsnap print and vxsnap list commands.

Controlling instant snapshot synchronization

Note: Synchronization of the contents of a snapshot with its original volume is not possible for space-optimized instant snapshots.

The commands in this section cannot be used to control the synchronization of linked break-off snapshots.

By default, synchronization is enabled for the `vxsnap refresh` and `restore` operation on instant snapshots. Otherwise, synchronization is disabled unless you specify the `sync=yes` attribute to the `vxsnap` command.

The following table shows the commands that are provided for controlling the synchronization manually.

Command	Description
`vxsnap [-g diskgroup] syncpause vol`	Pause synchronization of a volume.
`vxsnap [-g diskgroup] syncresume vol`	Resume synchronization of a volume.
`vxsnap [-b] [-g diskgroup] syncstart vol`	Start synchronization of a volume. The `-b` option puts the operation in the background.
`vxsnap [-g diskgroup] syncstop vol`	Stop synchronization of a volume.
`vxsnap [-g diskgroup] syncwait vol`	Exit when synchronization of a volume is complete. An error is returned if *vol* is invalid (for example, it is a space-optimized snapshot), or if *vol* is not being synchronized.

The `vxsnap snapwait` command is provided to wait for newly linked break-off snapshot volumes or volume sets to become ACTIVE, or for reattached snapshot volumes to reach the SNAPDONE state following resynchronization.

See “Creating and managing linked break-off snapshot volumes” on page 108.

See “Reattaching a linked break-off snapshot volume” on page 112.

Improving the performance of snapshot synchronization

Two optional arguments to the `-o` option are provided to help optimize the performance of synchronization when using the `make`, `refresh`, `restore` and `syncstart` operations:

`iosize=`*`size`* — Specifies the size of each I/O request that is used when synchronizing the regions of a volume. Specifying a larger size causes synchronization to complete sooner, but with greater impact on the performance of other processes that are accessing the volume. The default *size* of 1m (1MB) is suggested as the minimum value for high-performance array and controller hardware. The specified value is rounded to a multiple of the volume's region size.

`slow=`*`iodelay`* — Specifies the delay in milliseconds between synchronizing successive sets of regions as specified by the value of `iosize`. This can be used to change the impact of synchronization on system performance. The default value of iodelay is 0 milliseconds (no delay). Increasing this value slows down synchronization, and reduces the competition for I/O bandwidth with other processes that may be accessing the volume.

Options may be combined as shown in the following examples:

```
# vxsnap -g mydg -o iosize=2m,slow=100 make \
  source=myvol/snapvol=snap2myvol/syncing=on

# vxsnap -g mydg -o iosize=10m,slow=250 syncstart snap2myvol
```

Note: These optional parameters only affect the synchronization of full-sized instant snapshots. They are not supported for space-optimized snapshots.

Chapter 6

Administering volume templates and other configuration elements

The Veritas Intelligent Storage Provisioning (ISP) feature creates volumes with a set of rules or capabilities. A volume template (or template for short) is a collection of rules that provide capabilities. ISP refers to templates when creating a volume, and selects an appropriate set of templates to use based on the requested capabilities.

ISP follows certain rules when selecting templates to match a requested set of capabilities.

See “Volume templates” on page 140.

The procedures in this chapter use the `vxtemplate` command. For full information about using this command, see the `vxtemplate`(1M) manual page.

Installing configuration elements in the ISP database

Storage pool, storage pool set, template set, template and capability definitions are referred to as *configuration elements*. A standard set of configuration elements are included in the Configuration Database that is installed with the ISP package.

These configuration elements cover a broad range of uses, and should be sufficient for most applications. To make additional configuration elements available for use, they must be installed in the ISP Configuration Database.

To install additional elements that are defined in a file, use the following command:

```
# vxtemplate -C -d element_defs_file install
```

You can select only to install selected elements from a file by naming them explicitly as shown here:

```
# vxtemplate -r -C -d my_templates install template=Reliability
```

The `-r` (recursive) option ensures that all capabilities and templates that are referred to by the `Reliability` template are also installed.

See "ISP configuration elements" on page 177.

Installing configuration elements in storage pools and disk groups

If you specify a template set, storage pool definition or a storage pool set definition when creating a storage pool, the required templates and capabilities are automatically associated with the storage pool and with its disk group.

See "Creating a storage pool" on page 52.

A storage pool's `selfsufficient` policy governs whether configuration elements can be installed automatically from the ISP Configuration Database as required. If the policy is set to `host`, elements are available to the pool from the ISP Configuration Database, from the disk group and from the pool itself. If the policy is set to `diskgroup`, the pool can install additional elements from the disk group, but not from the ISP Configuration Database. A policy value of `pool` means that only elements that are installed in the pool can be used. If required, you can use the `vxtemplate` command to install templates, capabilities and template sets from the ISP Configuration Database into disk groups and pools.

Template sets are convenient for installing many capabilities and templates in a single operation. The following example shows how to install all the capabilities and templates from the template set, `DataMirroring`, into a storage pool, `mypool`, and its disk group, `mydg`:

```
# vxtemplate -g mydg -P mypool install
  template_set=DataMirroring
```

This is equivalent to using this (very long) command:

```
# vxtemplate -r -g mydg -P mypool install \
  template=DataMirroring,ArrayProductId,\
ConfineLogsToSimilarStorage,\
ConfineMirrorsToSimilarStorage,DCOLogMirroring,\
InstantSnapshottable,LogsOnSeparateComponents,\
MirrorsOnSeparateComponents
```

Listing and printing configuration elements

Use the following command to list the templates and capabilities that are associated with a specified disk group and/or storage pool.

```
# vxtemplate [-g diskgroup] [-P pool] list
```

This command displays results similar to the following.

```
TY  NAME                          STATE     AUTOINSTALL
vt  Mirroring                     Active    0
vt  MirroringStriping             Active    0
vt  Raid5Template                 Active    0
vt  Striping                      Active    0
cp  ParityReliable                -         -
cp  PerformanceByStriping         -         -
cp  Reliability                   -         -
```

Here the types `vt` and `cp` indicate volume template and capability definitions.

To print the details of these templates and capabilities, use this command:

```
# vxtemplate [-g diskgroup] [-P pool] print
```

You can print the details of only certain elements by specifying these as arguments to the command:

```
# vxtemplate [-g diskgroup] [-P pool] print \
  [template=t1[,t2...]] [capability=c1[,c2...]] \
  [template_set=ts1[,ts2...]] [storage_pool_set=ps1[,ps2...]] \
  [pool_definition=pd1[,pd2...]]
```

For example, to print the definitions of the `Mirroring` and `Striping` volume templates that are associated with the storage pool, `mypool`, in the disk group, `mydg`, you would use this command:

```
# vxtemplate -g mydg -P mypool print template=Mirroring,Striping
```

The following command lists all the templates and capabilities that are available in the ISP Configuration Database on the system:

```
# vxtemplate -C list
```

The output from this command is similar to the following.

```
TY  NAME                          STATE    AUTOINSTALL
vt  DataMirroring                 -          -
vt  Raid5Volume                   -          -
vt  Reliability                   -          -
vt  Striping                      -          -
cp  DataMirroring                 -          -
cp  DataRedundancy                -          -
cp  Raid5LogMirroring             -          -
cp  Raid5Capability               -          -
cp  Reliability                   -          -
cp  Striping                      -          -
ts  Raid5Templates                -          -
pd  raid5_volumes                 -          -
ps  mirrored_data_striped_clones  -          -
...
```

Here the types `ts`, `pd` and `ps` indicate template set, storage pool and storage pool set definitions respectively.

To print the details of elements in the Configuration Database, use this command:

```
# vxtemplate -C print [template=t1[,t2...]] \
  [capability=c1[,c2...]] [template_set=ts1[,ts2...]] \
  [storage_pool_set=ps1[,ps2...]] \
  [pool_definition=pd1[,pd2...]]
```

As for storage pools and disk groups, you can print the details of certain elements by specifying these as arguments to the command, for example:

```
# vxtemplate -C print template_set=DataMirroring,Raid5Templates
```

You can also use the `listcapability`, `listtemplate`, `listtemplateset`, `listpooldefinition` and `listpoolset` keywords of `vxtemplate` to list only the capabilities or templates that are associated with the specified disk group and/or storage pool. Similarly, the `printcapability`, `printtemplate`, `printtemplateset`, `printpooldefinition` and `printpoolset` keywords are used to print the element definitions in detail. For more information, see the `vxtemplate`(1M) manual page.

Deactivating and reactivating templates

By default, templates are activated when they are associated with a storage pool. You can run the `vxtemplate deactivate` command to prevent one or more templates from being used in the storage pool with which they have been associated:

```
# vxtemplate [-g diskgroup] [-P pool] deactivate \
  template=t1[,t2...]
```

If the template is currently in use, ISP displays an error message.

You can reactivate templates by using the `vxtemplate activate` command:

```
# vxtemplate [-g diskgroup] [-P pool] activate \
  template=t1[,t2...]
```

Renaming a capability

To change the name of a capability, use the following command:

```
# vxtemplate [-C | -g diskgroup] renamecapability \
  oldname newname
```

For example, you may want to change the name if you have modified the attributes of the capability for a particular disk group.

Renaming a template

To change the name of a template, use the following command:

```
# vxtemplate [-C | -g diskgroup] renametemplate \
  oldname newname
```

For example, you may want to change the name if you have modified a template for a particular disk group.

Finding templates for specified capabilities

To list the template definitions that implement certain capabilities, use the following command:

```
# vxtemplate [-C | -g diskgroup] list \
  [capability=c1[,c2...]]
```

For example, to list all the templates in the ISP Configuration Database that implement the `DataMirroring` capability:

```
# vxtemplate -C list capability=DataMirroring
TY NAME
vt DataMirroring
vt MultipathingThroughMirroring
```

If no capabilities are specified, the command lists all templates that are defined in the ISP Configuration Database or specified disk group.

Listing template dependencies

Volume template definitions can extend or apply other volume template definitions. To list the templates that depend on one or more given templates, use the following command:

```
# vxtemplate [-C | -g diskgroup] -w listtemplate \
  template1 [template2 ...]
```

To list all the templates upon which one or more given templates depend, use the following command:

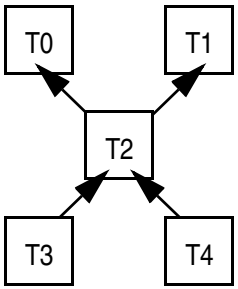
```
# vxtemplate [-C | -g diskgroup] -r listtemplate \
  template1 [template2 ...]
```

If the `-C` option is specified, the dependencies in the ISP Configuration Database are displayed. If the `-g` option is used, only the dependencies for the templates that are currently installed in the specified disk group are shown.

The `-w` (whole) and `-r` (recursive) options can be specified together to list all templates that are linked by inheritance.

Figure 6-13 shows an example hierarchy of template dependencies. The arrows show the direction of inheritance.

Figure 6-13 Example template dependencies



The following command examples show the output that would be returned for this hierarchy:

```
# vxtemplate -g mydg -w listtemplate T0
T0
# vxtemplate -g mydg -w listtemplate T2
T0
T1
T2
# vxtemplate -g mydg -w listtemplate T3
T0
T1
T2
T3
# vxtemplate -g mydg -r listtemplate T1
T2
T3
T4
# vxtemplate -g mydg -w -r listtemplate T2
T0
T1
T2
T3
T4
```

Note: A template is always shown as depending on itself as it is required to make itself "whole."

The next example returns the templates on which the `PrefabricatedRaid5` template in the ISP Configuration Database depends:

```
# vxtemplate -C -r listtemplate PrefabricatedRaid5
TY NAME                    STATE              AUTOINSTALL
vt ArrayPrductId           -                  -
```

Listing capability dependencies

Capability definitions can extend or apply other capability definitions. To list the capabilities that depend on one or more given capabilities, use the following command:

```
# vxtemplate [-C | -g diskgroup] -w listcapability \
  capability1 [capability2 ...]
```

To list all the capabilities upon which one or more given capabilities depend, use the following command:

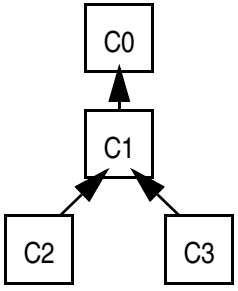
```
# vxtemplate [-C | -g diskgroup] -r listcapability \
  capability1 [capability2 ...]
```

If the `-C` option is specified, the dependencies in the ISP Configuration Database are displayed. If the `-g` option is used, only the dependencies for the capabilities that are currently installed in the specified disk group are shown.

The `-w` (whole) and `-r` (recursive) options can be specified together to list all capabilities that are linked by inheritance.

Figure 6-14 shows an example hierarchy of capability dependencies. The arrows show the direction of inheritance.

Figure 6-14 Example capability dependencies



The following command examples show the output that would be returned for this hierarchy:

```
# vxtemplate -g mydg -w listcapability C0
C0
# vxtemplate -g mydg -w capability listcapability C1
C0
C1
# vxtemplate -g mydg -w capability listcapability C2
C0
C1
C2
# vxtemplate -g mydg -r capability listcapability C1
C2
```

```
C3
# vxtemplate -g mydg -w -r listcapability C1
C0
C1
C2
C3
```

Note: A capability is always shown as depending on itself as it is required to make itself "whole."

The next example returns the capabilities which depend on the `DataRedundancy` capability in the ISP Configuration Database:

```
# vxtemplate -C -w listcapability PrefabricatedRaid5
TY NAME                 STATE            AUTOINSTALL
cp DataMirroring        -                -
cp DataMirrorStripe     -                -
cp DataRedundancy       -                -
cp DataStripeMirror     -                -
cp Raid5Capability      -                -
```

Listing template sets

To list the template sets in the ISP Configuration Database that contain certain template definitions, use the following command:

```
# vxtemplate -C listtemplateset [template=t1[,t2...]]
```

For example, to list all the template sets that contain the template `DataMirroring`:

```
# vxtemplate -C listtemplateset template=DataMirroring
TY NAME
ts DataMirroring
ts DataMirroringPrefabricatedRaid5
ts DataMirroringPrefabricatedStriping
```

If no templates are specified, the command lists all template sets that are defined in the ISP Configuration Database.

Listing templates included by template sets

To list the templates that are included in certain template sets in the ISP Configuration Database, use the following command:

```
# vxtemplate -C listtemplate [template_set=ts1[,ts2...]]
```

For example, to list all the templates that are included in the `ConcatVolumes` template set:

```
# vxtemplate -C listtemplate template_set=ConcatVolumes
TY NAME                                STATE       AUTOINSTALL
vt ArrayProductID                      -           -
vt ConcatVolumes                       -           -
vt ConfineLogsToSimilarSTorage         -           -
vt DCOLogMirroring                     -           -
vt DCOLogStriping                      -           -
vt InstantSnapshottable                -           -
vt LogsOnSeparateComponents            -           -
```

If no template sets are specified, the command lists the templates that are included in all template sets in the ISP Configuration Database.

Finding template sets containing specified templates

To list the template set definitions in the ISP Configuration Database that contain certain template definitions, use the following command:

```
# vxtemplate -C listtemplateset [template=t1[,t2...]]
```

For example, to list all the template sets that contain the template `DataMirroring`:

```
# vxtemplate -C listtemplateset template=DataMirroring
TY NAME
ts DataMirroring
ts DataMirroringPrefabricatedRaid5
ts DataMirroringPrefabricatedStriping
```

If no templates are specified, the command lists all template sets that are defined in the ISP Configuration Database.

Listing templates included by storage pool definitions

To list the templates that are included in certain storage pool definitions in the ISP Configuration Database, use the following command:

```
# vxtemplate -C listtemplate [pooldefn=pd1[,pd2...]]
```

For example, to list all the templates that are included in the `striped_volumes` storage pool definition:

```
# vxtemplate -C listtemplate pooldefn=striped_volumes
TY NAME                                STATE       AUTOINSTALL
vt ArrayProductID                      -           -
vt ColumnsOnSeparateComponents         -           -
vt ConfineColumnToSimilarStorage       -           -
vt ConfineToSimilarStorage             -           -
vt ConfineToSpecificStorage            -           -
vt DCOLogStriping                      -           -
vt ExcludeSpecificStorage              -           -
vt InstantSnapshottable                -           -
vt MultipathingThroughMultiplePaths    -           -
vt Striping                            -           -
```

If no storage pool definitions are specified, the command lists the templates that are included in all storage pool definitions in the ISP Configuration Database.

Removing templates, capabilities and template sets

To remove a template, capability or template set from a disk group and/or storage pool, or from the ISP Configuration Database, use the `vxtemplate uninstall` command. For example, the following command removes the definitions of the DataMirroring template and capability from the storage pool, `mypool`, in the disk group, `mydg`:

```
# vxtemplate -g mydg -P mypool uninstall \
  DataMirroring capability=DataMirroring
```

The following command similarly removes this template and capability from the ISP Configuration Database:

```
# vxtemplate -C uninstall DataMirroring \
  capability=DataMirroring
```

Note: If a template is associated with any ISP volumes, you cannot dissociate the template from a storage pool, or uninstall the template from a disk group without first deleting the volumes. To display which templates are associated with ISP volumes, use the following command:

```
# vxassist -g diskgroup printintent volume ...
```

Chapter 7

Creating and modifying user templates

When using Veritas Intelligent Storage Provisioning (ISP) to create an application volume with a specified set of capabilities, you can usually enter values for the parameters that modify these capabilities. For example, you can select the degree of redundancy, fault tolerance, or performance that you require from the volume. For convenience in creating volumes, you can set up user templates to store commonly used sets of capabilities and parameter values. You can then specify the name of a user template instead of the name of a capability when using the `vxassist` command to create an application volume.

Format of user templates

A user template consists of a list of capabilities and rules. Associated with each capability is a list of parameter-value pairs that define default parameter values for the capability. The format of a typical user template specification is shown here:

```
user_template usertmplt_tname {
    description "string"
    descriptionid id
    capability capability1 {
        attribute_name:type = default_value
        ...
    }
    capability capability2 {
        attribute_name = default_value
        ...
    }
    ...
    rules {
        [tag "volume_tag"]
        rule1
```

```
            rule2
            ...
        }
    };
```

This definition has the following important components:

usertmplt_name	Defines the name of the user template, for example, `MyReliableReplication`, and `MySWSnapshot`.
`description`	Describes a user template in English.
`descriptionid`	References the message catalog where localized versions of the description may be found, and provides an index number for the description in the catalog.
`capability`	Indicates the capabilities that are provided by volumes created from the user template.
attribute_name	Defines the name and value of a parameter.
`rules`	Specifies the start of the section that contains the rules for selecting and laying out storage, and which implement the capabilities that the user template provides. See “Rules” on page 144.
`tag`	Specifies an optional default tag for a volume.

See “User template” on page 169.

For example, consider a capability named `HardwareReplication` that has two variable parameters that specify the source and the destination:

```
capability HardwareReplication {
    var source:string
    var destination:string
};
```

In addition, the capability named `Reliable`, provides reliability by creating a mirrored volume:

```
capability Reliable {
    var NMIRS:int
};
```

Rules can also be specified to user templates, as shown in these examples that could be used for creating application volumes for database tables and indexes:

```
user_template DB_Table {
    description "Makes mirrored volume for a database table"
    capability DataMirroring {
        NMIRS = 2
    }
    rules {
        confineto "VendorName"="EMC"
    }
};

user_template DB_Index {
    description "Makes high performance volume for a database
index"
    capability Striping {
        NCOLS = 8
    }
    rules {
        confineto "VendorName"="EMC"
    }
};
```

The user template, `DB_Table`, can be used to create a mirrored volume with two plexes, but is restricted to using only EMC disks. Similarly, the user template, `DB_Index`, can be used to create a striped volume with eight columns, and is also restricted to using only EMC disks.

Creating user templates

You can either create a user template by defining it directly from the command line, or by defining it in a configuration file. The first method is only suitable for very simple user templates. It is usually preferable to define your templates in a configuration file.

To create a user template directly from the command line, use the `vxusertemplate create` command as shown here:

```
# vxusertemplate create usertemplate_name \
  [description="string"] \
  [rules=rule1 [rule2 ]...] \
  [capability=capability1[(var1=value1[,var2=value2]...)]\
  [,capability2[(var1=value1[,var2=value2]...)]]...]
```

For example, the following command creates a user template named `RP_DB_Table` that has both `Reliable` and `HighPerformance` capabilities:

```
# vxusertemplate create RP_DB_Table \
  description="Makes reliable high performance volume \
  for database table"rules=confineto "VendorName"="EMC" \
  capability='DataMirroring(NMIRS=2),Striping(NCOLS=8)'
```

This command adds the user template to the global `usertemplates` file that is maintained by ISP.

Alternatively, you can create a configuration file that contains an equivalent user template, as shown in this sample listing:

```
user_template DB_Table {
    description "Makes mirrored volume for database table"
    capability DataMirroring {
        NMIRS = 2
    }
    rules {
        confineto "VendorName"="EMC"
    }
};

user_template DB_Index {
    description "Makes high performance volume for database
index"
    capability Striping {
        NCOLS = 8
    }
    rules {
        confineto "VendorName"="EMC"
    }
};
```

You can then use the `-d` option with the `vxusertemplate create` command to add the user template in the configuration file to the global `usertemplates` file, as shown here:

```
# vxusertemplate -d config_file create
```

Using a user template to create an application volume

You can create an application volume using a user template by specifying the name of the user template in place of the name of a capability to the `vxassist make` command. For example, to create a 10GB application volume in the storage pool, `mypool`, using the user template, `RP_DB_Table`, you would use the following command:

```
# vxassist -g mydg -P mypool make myDBvol 10g \
  user_template='RP_DB_Table'
```

Note: Unlike volume templates, you cannot specify parameter values for user templates on the command line.

Listing currently defined user templates

Use the following command to list all the user templates that are currently defined in the global `usertemplates` file:

```
# vxusertemplate list
```

Printing user template definitions

Use the following command to print detailed information about one or more user templates that are currently defined in the global `usertemplates` file:

```
# vxusertemplate print usertemplate1 [usertemplate2 ...]
```

Deleting user templates

To remove a user template definition from the global `usertemplates` file, use the following command:

```
# vxusertemplate delete usertemplate
```

Chapter 8

Using capabilities, templates and rules

Veritas Intelligent Storage Provisioning (ISP) provides a structured and flexible rule-based declarative language for expressing how an application volume with a given set of characteristics is to be configured from the available storage. The language can be used to specify rules for allocating storage to the individual parts of a volume, such as mirrors, columns, logs and snapshots, as well as to an entire volume.

See the following sections in this chapter for more information:

- Capabilities
- Volume templates
- Rules
- Storage selection rules
- Storage selection rule operators
- Storage layout rules
- Compound rules
- Attribute aliases
- Simplified syntax for rules on the command line

See "ISP language definition" on page 159.

See "ISP configuration elements" on page 177.

Capabilities

A capability, as expressed in the ISP language, is a description of what a volume is capable of doing. Each template is associated with one or more capabilities so that ISP can choose those templates that are suitable for use in allocating storage to volumes. A capability is interpreted by ISP as a tag.

The format of a typical capability definition is shown here:

```
capability cap_name {
    extends capname1,capname2, ...
    description text
    descriptionid id
    display_name name
    display_name_id catalogid, index
    var varname1:type {
        defaultvalue value
        description description
        descriptionid catalogid, index
    }
    var varname2:type {
        defaultvalue value
        description text
        descriptionid catalogid, index
    }
    ...
};
```

This definition has the following important components:

cap_name	Indicates the nature of the capability, for example, `Reliable`, `Performant`, `Snapshot` and `EMCSnapshot`.
`extends`	Indicates that the capability is derived from other named capabilities. See “Inheritance of capabilities” on page 139.
`description`	Describes a capability or one of its variable parameters in English.
`descriptionid`	References the message catalog where localized versions of the description may be found, and provides an index number for the description in the catalog.
`display_name`	Defines the English name of the capability that is displayed in the VEA.
`display_name_id`	References the message catalog where localized versions of the display name may be found, and provides an index number for the display name in the catalog.

`var`	Defines a variable that may be specified as a parameter and its value to a capability. A template can use the variables in its rules section to provide the requested capability. Permitted variable types are `int` or `string`.
`defaultvalue`	Defines the default value of a capability's parameter if no value is specified.

See "Capabilities" on page 138.

The following is a sample capability definition:

```
capability DataMirroring {
    display_name "Data Mirroring"
    display_name_id "{b84f1c64-1dd1-11b2-8b42-080020feef8b}", 3

    description "Volume has multiple copies of data."
    descriptionid "{b84f1c64-1dd1-11b2-8b42-080020feef8b}", 4

    extends DataRedundancy
    var nmirs:int {
        display_name "Number of data copies"
        display_name_id "{b84f1c64-1dd1-11b2-8b42-
        080020feef8b}", 5

        description "Specify the number of copies of data."
        descriptionid "{b84f1c64-1dd1-11b2-8b42-080020feef8b}",
        6

        defaultvalue 2
    }
};
```

Inheritance of capabilities

The `extends` keyword in a capability definition allows a capability to inherit some of the properties of one or more *base* capabilities. This implies that such a *derived* capability has an "is a" relationship with these base capabilities. The following rules are applied by the inheritance mechanism:

- A derived capability inherits variables from all its base capabilities.
- A derived capability cannot define variables with the same name as any of the variables in any of its base capabilities.
- If a capability is derived from more than one base capability, none of the variable names in any of its base capabilities should be the same as a variable name in any other base capability.

The following sample capability definitions show how the `DataMirrorStripe` capability is derived from the more general `DataMirroring` and `Striping` capabilities:

```
capability DataMirrorStripe {
```

```
        extends DataMirroring, Striping
    };
```

`DataMirrorStripe` also inherits the variables `nmirs` and `ncols` from the two capabilities that it extends.

Volume templates

A volume template (or template for short) is a meaningful collection of rules that provides capabilities for volumes as defined by those rules, or by reference to other capabilities. ISP refers to templates when creating an application volume. It chooses an appropriate template to use based on the capabilities that you request for the volume, and allocates storage by following the rules that the template contains.

The format of a typical template specification is shown here:

```
volume_template template_name {
    extends template1, template2, ...
    inherits capability5, capability6, ...
    provides capability1, capability2, ...
    requires capability3, capability4, ...
    description text
    descriptionid id
    rules {
        [tag "volume_tag"]
        rule1
        rule2
        ...
    }
};
```

This definition has the following important components:

template_name	Indicates the nature of the volume that can be created by the template, for example, `DataMirroring`, `Raid5Volume` and `InstantSnapshottable`.
`extends`	Indicates that the template includes all the capabilities, rules and variables from the named templates. See “extends” on page 141.
`inherits`	Indicates that the template can inherit capabilities when required from those listed. See “inherits” on page 142.
`provides`	Indicates the capabilities that are provided by volumes created from the template. See “provides” on page 143.
`requires`	Indicates that the template can acquire rules when required from other templates that provide the specified

	capabilities. See “requires” on page 143.
`description`	Describes a template in English.
`descriptionid`	References the message catalog where localized versions of the description may be found, and provides an index number for the description in the catalog.
`rules`	Specifies the start of the section that contains the rules for selecting and laying out storage, and which implement the capabilities that the template provides. The rules usually operate on the values that were specified for the arguments of the requested capabilities. See “Rules” on page 144. See “Compound rules” on page 154.
`tag`	Specifies an optional default tag for a volume.

The following sections describe the keywords in detail:

- `extends`
- `inherits`
- `provides`
- `requires`

See “Volume template” on page 163.

extends

A template can derive some of its properties by extending one or more base templates as shown in this example:

```
volume_template DerivedTemplate {
    extends BaseTemplate
    rules {
        ...
    }
};
```

By extending template `BaseTemplate`, the template `DerivedTemplate`:

- Provides all capabilities that `BaseTemplate` provides.
  See “provides” on page 143.
- Requires all capabilities that `BaseTemplate` requires.
  See “requires” on page 143.
- Inherits all capabilities that `BaseTemplate` inherits.
  See “inherits” on page 142.
- Obtains and applies all the rules of `BaseTemplate`.

- Can use variables of all the capabilities that `BaseTemplate` provides.

Note: `BaseTemplate` and `DerivedTemplate` cannot use different capabilities that have variables with the same name.

The derived template has an “is a” relationship with its base templates. The derived template can be used instead of any of its base templates.

The following example demonstrates the application of the `extends` keyword:

- Extend `ReliablePerformant` into the template `MyReliablePerformant`:

```
volume_template MyReliablePerformant {
    extends ReliablePerformant
    rules {
        ...
    }
};
```

inherits

If a template specifies that it can inherit a capability, it provides that capability when required. Inheriting a capability is equivalent to the combination of requiring and providing a capability.

See “requires” on page 143.

See “provides” on page 143.

The following example demonstrates the application of the `inherits` keyword:

- The following template definition:

```
volume_template ReliableSnapshot {
    requires Reliable
    provides Reliable, Snapshot
    rules {
        ...
    }
};
```

  can be rewritten as:

```
volume_template ReliableSnapshot {
    inherits Reliable
    provides Snapshot
    rules {
        ...
    }
};
```

provides

A template provides one or more capabilities as defined by its rules, or by requiring capabilities.

See “requires” on page 143.

Any variables that are defined for the capability can be used by the template’s rules.

The following example demonstrates the application of the `provides` keyword:

- Define a template, `ReliableT`, that provides the `Reliable` capability, and which uses the `NMIRS` variable of the capability to set the number of mirrors:

```
volume_template ReliableT {
    provides Reliable
    rules {
        mirror NMIRS {
            ...
        }
    }
};
```

requires

If a template requires one or more capabilities, it can take the appropriate rules from any templates that provide those capabilities.

The `requires` keyword does not imply `provides`. Whenever a template provides a capability by specifying that it requires it, it should be explicitly listed in a `provides` or `inherits` clause.

See “provides” on page 143.

See “inherits” on page 142.

The following example demonstrates the application of the `requires` keyword:

- Define a template, `MySnapshot`, that provides the SnapShot capability, and requires, but does not provide, the Reliable capability:

```
volume_template MySnapshot {
    requires Reliable
    provides Snapshot
    rules {
        ...
    }
};
```

  This template picks up any templates that provide the `Reliable` capability, and merges their rules with its own rules. If the template were

also required to provide the `Reliable` capability, the `inherits` keyword should be used as shown here:

```
volume_template MyReliableSnapshot {
    inherits Reliable
    provides Snapshot
    rules {
        ...
    }
};
```

This template provides the Snapshot capability through its own rules, and also provides the `Reliable` capability by using rules from other templates.

Note: By allowing ISP to choose any template that provides the required capability, the `requires` keyword gives ISP more flexibility when allocating storage. This behavior is known as *dynamic inheritance*.

When choosing a template with a required capability, ISP gives preference to templates that provide only that capability. This behavior avoids giving unrelated capabilities to a volume.

When searching for a template with a required capability, ISP chooses only those templates that inherit or provide that capability.

When searching for a template with a required capability, ISP recursively chooses templates that provide all the required capabilities.

Rules

Rules specify the criteria for allocating storage, the criteria for laying out VxVM objects on storage, and the relationship between these objects. Rules can be classified as storage selection rules that define how to choose storage, or as storage layout rules that define how storage is to be used.

You can optionally precede any rule with the keyword `desired`. This modifier means that ISP tries to honor this rule, but it can discard it if the rule would cause the allocation to fail. An optional integer after the `desired` keyword specifies the preference order value for the rule. If specified, this number must be greater than or equal to 1. If no preference order value is specified for a rule, a value of 1 is assumed. Rules with low preference order values are preferred to be retained over rules with higher values. Rules with the same preference order are discarded together.

In the following example, the `stripe` rule can be dropped if there is insufficient storage to fulfill the storage allocation request:

```
mirror 2 {
    desired 10 confineto "enclosure"
    desired 15 stripe 15
}
```

Storage selection rules

The following sections describe storage selection rules:

- affinity
- confineto
- exclude
- multipath
- select
- separateby
- strong separateby

You can limit the scope of interpretation of rules to the components of an application volume, such as its plexes, columns and logs.

See "Compound rules" on page 154.

Multiple arguments to the `confineto`, `exclude` and `select` storage selection rules can be combined by using operators. If an operator is not specified, ISP uses the default operator for the rule.

See "Storage selection rule operators" on page 151.

Note: The `=`, `!=`, `<`, `<=`, `>` and `>=` comparative operators can be used with the `confineto`, `exclude` and `select` storage selection rules. However, the operand arguments to the `<`, `<=`, `>` and `>=` operators must be signed or unsigned integers that are capable of being represented by 64 or fewer bits. If the operands are character strings, real numbers, or are integers longer than 64 bits, the result of the comparison may be incorrect.

---

affinity

An `affinity` rule expresses attraction between VxVM objects. Objects that conform to this rule share as many attribute values as possible.

Note: The `affinity` rule does not take any rule operators.

The following example demonstrates the application of `affinity` rules:

- Allocate storage from as few enclosures as possible:
  ```
  desired affinity "Enclosure"
  ```

confineto

A `confineto` rule restricts a VxVM object, such as volume or mirror, to being configured from a specific set of LUNs. The scope of the rule determines the VxVM object for which the restrictions apply. When a `confineto` rule is used at the top level, it usually applies to the volume.

You can use compound rules to confine a mirror, column or log to a set of LUNs.

See "Compound rules" on page 154.

Note: Expressions involving `confineto` can use the `allof`, `anyof`, `eachof`, `noneof` and `oneof` operators to combine multiple arguments. By default, ISP applies the `eachof` operator.

See "Storage selection rule operators" on page 151.

The expression argument of a `confineto` rule usually consists of one or more LUN attributes that can be specified either with or without an accompanying value. The LUN attributes can be auto-discovered or user-defined. A value that is not quoted is interpreted as the name of a variable whose value is to be determined when a VxVM object is created.

The following examples demonstrate the application of `confineto` rules:

- Use storage having the same value for the user-defined attribute, `Room`:
  ```
  confineto "Room"
  ```
- By default, the `eachof` operator is assumed for a `confineto` rule. The following rules, which specify that storage is only to be assigned from EMC LUNs or from LUNs that share the same value for the `Room` attribute, are equivalent:
  ```
  confineto "Room", "VendorName"="EMC"
  confineto eachof("Room", "VendorName"="EMC")
  ```

- Use only EMC LUNs that have their Room attribute set to the value Room1:
```
confineto eachof("VendorName"="EMC", "Room"="Room1")
```
- Use only storage with VendorName set to the value of the variable VENDOR_NAME, and with Room set to the value of the variable ROOM_NAME:
```
confineto eachof("VendorName"=VENDOR_NAME, "Room"=ROOM_NAME)
```

Note: VENDOR_NAME and ROOM_NAME are variables that are defined in a capability. Their values are resolved when you enter values for the capability during the creation of a VxVM object.

- Create a volume using one or more LUNs from either or both of the enclosures EMC1 and EMC2, and not from anywhere else:
```
confineto anyof("Enclosure"="EMC1", "Enclosure"="EMC2")
```

Note: When the anyof operator is used, ISP takes storage from the operands in the order that they are specified. In this example, ISP first attempts to select LUNs from the enclosure EMC1, and if this is not possible, it selects LUNs from the enclosure EMC2. If this also is not possible, ISP selects LUNs from both EMC1 and EMC2.

- Use LUNs that originate either only from EMC or only from Hitachi:
```
confineto oneof("VendorName"="EMC", "VendorName"="Hitachi")
```

Note: When the oneof operator is used, ISP takes storage from the operands in the order that they are specified. In this example, ISP first attempts to select EMC LUNs, and if this is not possible, it selects Hitachi LUNs.

- Confine storage to be allocated from either Room1 or from Room2. If storage is selected from Room1, Hitachi LUNs cannot be used. Similarly, if storage is selected from Room2, EMC LUNs cannot be used.
```
confineto
oneof(eachof("Room"="Room1",noneof("VendorName"="Hitachi")), \
            eachof("Room"="Room2", noneof("VendorName"="EMC")))
```

Note: In this example, ISP first attempts to select LUNs from Room1, and if this is not possible, it selects LUNs from Room2.

- Do not use LUNs from Room1 or LUNs from vendor EMC:
```
confineto noneof("VendorName"="EMC", "Room"="Room1")
```

Note: The noneof operator implies the logical union of its operands.

- Allocate storage from LUNs in `Room1` and `Room2`:
  ```
  confineto allof("Room"="Room1", "Room"="Room2")
  ```

  Note: In the absence of a specified operator, the `eachof` operator is assumed.

- Do not mix EMC LUNs with LUNs from other vendors when allocating storage:
  ```
  confineto oneof("VendorName"="EMC", noneof("VendorName"="EMC"))
  ```
- Allocate storage only from LUNs that have a `Columns` attribute value greater than 1, and a `Parity` attribute value of 0:
  ```
  confineto eachof("Columns">"1", "Parity"="0"))
  ```

exclude

An `exclude` rule omits a set of LUNs from being allocated to a VxVM object.

Note: Expressions involving `exclude` can only use the `allof` and `eachof` operators to combine multiple arguments. By default, ISP applies the `allof` operator.

See "Storage selection rule operators" on page 151.

The following examples demonstrate the application of `exclude` rules:

- The following rules, which prevent storage being assigned from EMC or Hitachi disks, are equivalent:
  ```
  exclude "VendorName"="EMC", "VendorName"="Hitachi"
  exclude allof("VendorName"="EMC", "VendorName"="Hitachi")
  ```
- Do not use EMC LUNs from `Room1`:
  ```
  exclude eachof("VendorName"="EMC", "Room"="Room1")
  ```
- Do not use disks with `VendorName` set to the value of the variable `VENDOR_NAME`:
  ```
  exclude "VendorName"=VENDOR_NAME
  ```

  Note: `VENDOR_NAME` is a variable that is defined in a capability. Its value is resolved when the VxVM object is created.

- Exclude certain LUNs from the enclosure Enclr1:
  ```
  exclude allof("DeviceName"="Enclr1_1", "DeviceName"="Enclr1_2")
  ```

multipath

A `multipath` rule specifies how tolerant a VxVM object is to the failure of a number of specified components. The rule defines how many paths a VxVM object should have available through each component.

Note: The `multipath` rule does not take any rule operators.

The following examples demonstrate the application of `multipath` rules:

- Tolerate the failure of one controller:
  ```
  multipath 2 "Controller"
  ```
- Tolerate the failure of one controller *and* one switch:
  ```
  multipath 2 "Controller", 2 "Switch"
  ```

select

A select rule specifies which storage to use for creating VxVM objects. When used outside of a sub clause, this rule is applied to an entire volume.

Note: Expressions involving `select` can use the `allof`, `anyof`, `eachof`, `noneof` and `oneof` operators to combine multiple arguments. By default, ISP applies the `anyof` operator.

See "Storage selection rule operators" on page 151.

The following examples demonstrate the application of `select` rules:

- The following rules, which try to allocate LUNs first from `Room1`, then from `Room2` if unsuccessful, and then from both locations, are equivalent:
  ```
  select "Room"="Room1", "Room"="Room2"
  select anyof("Room"="Room1", "Room"="Room2")
  ```

  Note: When the `anyof` operator is used, ISP takes storage from the operands in the order that they are specified.

- Use only EMC LUNs from `Room1`:
  ```
  select eachof("VendorName"="EMC", "Room"="Room1")
  ```

  Note: Here the `eachof` operator is used rather than the `allof` operator. The `eachof` operator implies the logical intersection of its operands. The `allof` operator implies the logical union of its operands.

- Use the specified LUNs from an enclosure:
  ```
  select "DeviceName"="Enclr1_1","DeviceName"="Enclr1_2"
  ```

separateby

A `separateby` rule is used to describe separation between VxVM objects. This is typically used to define failure domains to provide greater reliability by avoiding a single point of failure. For example, a `separateby` rule can be used to define that the mirrors of a volume should not share a controller. This makes the volume resilient to the failure of a controller.

Note: The `separateby` rule does not take any rule operators.

The following examples demonstrate the application of `separateby` rules:

- Allocate VxVM objects on separate enclosures:
  ```
  separateby "Enclosure"
  ```
- Allocate VxVM objects so that each object tolerates the failure of one controller and also uses LUNs from different manufacturers:
  ```
  separateby "VendorName", "Controller"
  ```

strong separateby

The `strong separateby` rule is a more restrictive form of `separateby` rule, which does not permit any sharing of attributes for the storage that is assigned to VxVM objects.

Note: The `strong separateby` rule does not take any rule operators.

The following example demonstrates the application of the `strong separateby` rule:

- Allocate VxVM objects so that each object is configured on a totally independent set of spindles.
  ```
  strong separateby "Spindles"
  ```

Storage selection rule operators

The following table lists the operators that are provided for combining the `confineto`, `exclude` and `select` storage selection rules:

Operator	Description
`allof`	Select available storage that matches the union of all the operands.
`anyof`	Select available storage that matches any of the operands in the order that they are listed. The first match is chosen.
`eachof`	Select available storage that matches each of the operands.
`noneof`	Do not select storage that matches any of the operands.
`oneof`	Select available storage that matches one and only one of the operands. The first match is chosen.

See the description of the individual selection rules for details of the applicability of these operators.

Storage layout rules

The following sections describe storage layout rules:

- `apply`
- `parity`
- `striped`

You can limit the scope of interpretation of rules to the components of an application volume, such as its plexes, columns and logs.

See “Compound rules” on page 154.

apply

The apply rule is used to apply rules from one or more specified templates in addition to the rules that appear in the current template. This rule acts recursively if the specified templates also use apply in their rules sections. The capabilities of any referenced templates are not used.

Expressions involving apply can specify the `eachof` or `oneof` operator. The default operator for this keyword is `eachof`.

The following examples demonstrate the application of the `apply` rule:

- Apply the rules but not the capabilities from the template `ArrayProductId` to a compound rule within the template `PrefabricatedDataMirroring`:

```
volume_template PrefabricatedDataMirroring {
    provides PrefabricatedDataMirroring
    rules {
        apply ArrayProductId
        confineto eachof ( "Redundancy" =nmirs , "Parity" ="0"  )
    }
};

volume_template ArrayProductId {
    provides ArrayProductId
    rules {
        confineto "ProductId"
    }
};
```

- Typically, you would use the apply keyword when you want to get the rules of a template without its associated capabilities. In the example, the template `MyReliableSnapshot` is defined so that the application of `Snapshot` to one mirror results in the whole volume getting the associated `Snapshot` capability, whereas the application of `EMCStorage` to one mirror does not result in the whole volume getting the `EMCStorage` capability:

```
volume_template MyReliableSnapshot {
    provides Snapshot, Reliable
    rules {
        separateby "Enclosure"
        mirror 1 {
            apply Snapshot, EMCStorage
            confineto "Enclosure"
        }
        mirror 1 {
            confineto "Enclosure"
        }
    }
};
```

parity

The `parity` rule defines whether redundancy should be parity based (that is, RAID-5) as implemented in either hardware or software. The value can be `true` or `false`.

The following example demonstrates the application of the `parity` rule:

- Define a template for providing reliability using software RAID-5:

```
volume_template Raid5Volume {
    provides Raid5Capability, Raid5LogMirroring
    rules {
        parity true
        stripe ncols - nmaxcols
        log nlogs {
            type raid5
        }
    }
};
```

striped

The striped rule indicates whether a volume or mirror is to be striped.

If `striped` is set to `true`, the number of columns is not necessarily greater than one as striping can be implemented in hardware as well as software.

The following example demonstrates the application of the `striped` rule:

- Define a template for providing reliability using mirrors, but where striping is not allowed:

```
volume_template ReliableConcat {
    provides Reliable, Concat
    rules {
        mirror NMIRS
        striped false
    }
};
```

Compound rules

Whether a rule applies to a volume, mirror, column or log depends on the scope in which the rule is applied. Compound rules are composed of several other rules, and are used to specify the scope of rules at a level below that of an entire volume. A compound rule can apply to one or more mirrors, columns or logs of a volume. It can also define how redundancy and separation are to be implemented for a volume.

Note: No more than two levels of nesting of compound rules may be specified.

The following sections describe the various types of compound rule:

- `log`
- `mirror`
- `mirror_group`
- `stripe`

mirror

The `mirror` rule is used to describe one or more mirrors of a volume. All the basic rules except `striped` and `parity`, and a restricted form of the `stripe` rule, can be used within a `mirror` rule. Rules that are specified within a `mirror` rule apply only to those mirrors that are constructed from this rule.

The following examples demonstrate the application of the `mirror` rule:

- Confine each mirror within an enclosure so that no mirror spans more than one enclosure:

```
mirror all {
    confineto "Enclosure"
}
```

- Configure two mirrors of a volume with the `EMCSnapshot` feature:

```
mirror 2 {
    apply EMCSnapshot
}
```

- Use `confineto` rules to force one mirror to be created on each of four separate enclosures:

```
mirror 1 {
    confineto "Enclosure"="enclr1"
}
mirror 1 {
    confineto "Enclosure"="enclr2"
}
mirror 1 {
    confineto "Enclosure"="enclr3"
}
mirror 1 {
    confineto "Enclosure"="enclr4"
}
```

mirror_group

The `mirror_group` rule groups together different mirrors of a volume. It should be used when there are groups of mirrors with several things that are common to each group, or when such groups need to have a `separateby` rule between them. Merging of mirrors can only take place within mirror groups.

The following example demonstrates the application of the `mirror_group` rule:

- Configure two mirror groups, `A` with two striped mirrors, and `B` with two concatenated mirrors, where the mirrors in each group lie within different enclosures:

```
mirror_group A {
    mirror 2 {
        confineto "Enclosure"
    }
    stripe 4
}

mirror_group B {
    mirror 2 {
        confineto "Enclosure"
    }
    striped false
}

mirror_group A,B {
    separateby "Enclosure"
}
```

stripe

The `stripe` rule describes one or more columns of a volume. All basic rules and some restricted form-related or mirror-related rules can be specified within a `stripe` rule. ISP merges multiple `stripe` rules to form a single `stripe` rule whose rules are the union of the separate `stripe` rules.

The following examples demonstrate the application of the `stripe` rule:

- Stripe a VxVM object over 5 columns:

  ```
  stripe 5
  ```

- Use `confineto` rules to force a column to be created on enclosures that are separate, but which are attached to the same controller, `ctlr1`:

  ```
  stripe 6-2 {
      confineto "Controller"="ctlr1"
      separateby "Enclosure"
  }
  ```

  Six columns are created by preference. If this is not possible, ISP attempts to create fewer columns down to a minimum of two.

log

The `log` rule describes a volume's logs. The argument of the rule specifies how many mirrors the log should have. Compound rules can include storage selection rules, stripe rules, and define the log type. The following log types may be defined:

`dco` Version 20 data change object (DCO) that can be used for both DRL and FastResync.

`raid5` RAID-5 log.

The following examples demonstrate the application of the `log` rule:

- Create a striped RAID-5 log with 4 columns, and a DCO log, each of which are confined to LUNs from a single (but possibly separate) enclosure:

  ```
  parity true
  log 1{
      type raid5
      confineto "Enclosure"
      stripe 4
  }
  log 1{
      type dco
      confineto "Enclosure"
  }
  ```

- Create a mirrored new-style DCO log, with each plex configured on a separate controller.

```
mirror 2
log 2{
    type dco
    separateby "Controller"
}
```

Attribute aliases

The following aliases are supported for attributes:

Attribute	Aliases	Description
`Controller`	`c, controller, cntrl, ctlr, ctrl`	The name of a controller, for example: `"Controller"="c1"`
`DA`	`d, da, daname, devicename`	A disk access name, for example: `"DA"="c0t4d0"`
`DM`	`disk, dm`	A disk media name, for example: `"DM"="mydg01"`
`Enclosure`	`enclosure, enclr`	The name of an enclosure, for example: `"Enclosure"="enc0"`
`LunId`	`lunid, udid`	A LUN identifier, for example: `"LunId"="0"`
`ProductId`	`pid, productid`	The product identifier for an array, for example: `"ProductId"="GR710"`
`RPM`	`diskrpm, rpm`	Disk rotation speed in revolutions per minute, for example: `"RPM"="7200"`
`Target`	`t, target`	A target identifier on a controller, for example: `"Target"="c0t4"`
`VendorName`	`vendor, vendorname`	The vendor identifier for an array, for example: `"VendorName"="FUJITSU"`

Simplified syntax for rules on the command line

For convenience, rules may be expressed in the following simplified form on the command line for the `vxassist` and `vxvoladm` commands:

```
rule_name=attribute_name[=value][,...]
```

The following examples of rule expressions:

```
confineto eachof("VendorName"="EMC","Room"="Room1","Building")
select anyof("Room"="Room1","Room"="Room2","Building")
exclude allof("VendorName"="EMC","VendorName"="Hitachi")
affinity "Enclosure"
separateby "Controller"
strong separateby "Spindles"
```

may be expressed by equivalent simplified rule expressions:

```
confineto=VendorName=EMC,Room=Room1,Building
select=Room=Room1,Room=Room2,Building
exclude=VendorName=EMC,VendorName=Hitachi
affinity=Enclosure
separateby=Controller
strong separateby=Spindles
```

The default storage selection rules are assumed, and cannot be substituted using the simplified form of the rule syntax.

Appendix A

ISP language definition

You can define the following objects using the Veritas Intelligent Storage Provisioning (ISP) specification language:

- Capability
- Volume template
- User template
- Storage pool
- Template set
- Storage pool set
- Volume group

Note the following points with regard to the syntax of the language:

- Each object category, such as capability, template, user template and storage pool and capability has its own name space. This means that a capability and a template can have the same name, but two capabilities cannot have the same name.
- All keywords are case insensitive.
- The primitive types are:

boolean	Possible values are **true** or **false**.
guid_string	String identifier for a GUI element.
int	Short signed integer.
long	Long signed integer.
quoted_string	String enclosed in double quotes.
string	String.
uint32	Unsigned 32-bit integer.

Syntax conventions

The following typographic conventions are used in the syntax description:

Typeface	Usage	Examples
monospace (bold)	Indicates a keyword.	**capability**
monospace (italic)	Indicates an irreducible term such as an identifier, and its type. A value is expected if only a type is specified.	*c_name*:*string* *uint32*
monospace	Indicates a reducible term.	capability_list

Symbol	Usage	Examples
[]	Indicates an optional syntax component.	[**extends** capability_list]
|	Separates mutually exclusive syntax components.	**dco** | **drl** | **raid5** | **srl**

Reserved keywords

The following keywords are reserved. They may not be used as identifiers for names of capabilities, templates and so on.

all	defaultvalue	int	selfsufficient
allof	description	log	separateby
affinity	descriptionid	mirror	server_plugin_id
aggregatable	desired	mirror_group	storage_pool
aggregate	display_name	multipath	storage_pool_set
anyof	display_name_id	none	string
application_template	dg	noneof	template
application_templates	domain	oneof	templates
apply	dontcare	parity	template_set
autogrow	eachof	pool	true
capabilities	exclude	pools	type
capability	extends	provides	user_template
clone	false	raid5	user_templates
column	feature	range	var
confineto	features	required	volume_template
const	group	requires	volume_templates
data	host	rules	widget_plugin_id
dcm	hosts	select	
dco	implements	stripe	
drl	inherits	strong	

Capability

```
capability c_name:string [ {
  [ extends string_list ]
  [ description quoted_string ]
  [ descriptionid msgcat_id:quoted_string, msg_id:int ]
  [ display_name quoted_string ]
  [ display_name_id msgcat_id:quoted_string, msg_id:int ]
```

```
  [ variable_list ]
} ] ;
```

See “Rules” on page 144.

string_list

```
string | string_list, string
```

variable_list

```
variable | variable_list variable
```

variable

```
var var_name:string : int | string [ {
  [ defaultvalue value:int | quoted_string ]
  [ description quoted_string ]
  [ descriptionid msgcat_id:quoted_string, msg_id:int ]
} ]
```

Volume template

```
volume_template vt_name:string {
  [ provides string_list ]
  [ requires string_list ]
  [ inherits string_list ]
  [ extends string_list ]
  [ description quoted_string ]
  [ descriptionid msgcat_id:quoted_string, msg_id:int ]
  [ rules [ {
        template_rules
  } ] ]
} ;
```

See “Volume templates” on page 140.

template_rules

```
[ [ desired [ pref_order ] ] confineto confineto_expr ] |
[ [ desired [ pref_order ] ] exclude exclude_expr ] |
[ [ desired [ pref_order ] ] select select_expr ] |
[ [ desired [ pref_order ] ] separateby string_list ] |
[ [ desired [ pref_order ] ] strong separateby string_list ] |
[ [ desired [ pref_order ] ] affinity string_list ] |
[ [ desired [ pref_order ] ] multipath multipath_expr ] |
[ [ desired [ pref_order ] ] striped boolean ] |
[ [ desired [ pref_order ] ] parity boolean ] |
[ [ desired [ pref_order ] ] apply string_list ] |
[ [ desired [ pref_order ] ] mirror int [ {
  [ [ desired [ pref_order ] ] confineto confineto_expr ]
  [ [ desired [ pref_order ] ] exclude exclude_expr ]
  [ [ desired [ pref_order ] ] select select_expr ]
  [ [ desired [ pref_order ] ] separateby string_list ]
  [ [ desired [ pref_order ] ] strong separateby string_list ]
  [ [ desired [ pref_order ] ] affinity string_list ]
  [ [ desired [ pref_order ] ] multipath multipath_expr ]
  [ [ desired [ pref_order ] ] striped boolean ]
  [ [ desired [ pref_order ] ] stripe from:int - to:int [ {
    [ [ desired [ pref_order ] ] confineto confineto_expr ]
    [ [ desired [ pref_order ] ] exclude exclude_expr ]
    [ [ desired [ pref_order ] ] select select_expr ]
    [ [ desired [ pref_order ] ] separateby string_list ]
    [ [ desired [ pref_order ] ] strong separateby string_list ]
    [ [ desired [ pref_order ] ] affinity string_list ]
  } ] ]
} ] ] |

[ [ desired [ pref_order ] ] mirror [ {
  [ [ desired [ pref_order ] ] confineto confineto_expr ]
  [ [ desired [ pref_order ] ] exclude exclude_expr ]
  [ [ desired [ pref_order ] ] select select_expr ]
```

```
  [ [ desired [ pref_order ] ] separateby string_list ]
  [ [ desired [ pref_order ] ] strong separateby string_list ]
  [ [ desired [ pref_order ] ] affinity string_list ]
  [ [ desired [ pref_order ] ] multipath multipath_expr ]
  [ [ desired [ pref_order ] ] striped boolean ]
  [ [ desired [ pref_order ] ] stripe from:int - to:int [ {
    [ [ desired [ pref_order ] ] confineto confineto_expr ]
    [ [ desired [ pref_order ] ] exclude exclude_expr ]
    [ [ desired [ pref_order ] ] select select_expr ]
    [ [ desired [ pref_order ] ] separateby string_list ]
    [ [ desired [ pref_order ] ] strong separateby string_list ]
    [ [ desired [ pref_order ] ] affinity string_list ]
  } ] ]
} ] ] |

[ [ desired [ pref_order ] ] mirror all [ {
  [ [ desired [ pref_order ] ] confineto confineto_expr ]
  [ [ desired [ pref_order ] ] exclude exclude_expr ]
  [ [ desired [ pref_order ] ] select select_expr ]
  [ [ desired [ pref_order ] ] separateby string_list ]
  [ [ desired [ pref_order ] ] strong separateby string_list ]
  [ [ desired [ pref_order ] ] affinity string_list ]
  [ [ desired [ pref_order ] ] multipath multipath_expr ]
  [ [ desired [ pref_order ] ] striped boolean ]
  [ [ desired [ pref_order ] ] stripe from:int - to:int [ {
    [ [ desired [ pref_order ] ] confineto confineto_expr ]
    [ [ desired [ pref_order ] ] exclude exclude_expr ]
    [ [ desired [ pref_order ] ] select select_expr ]
    [ [ desired [ pref_order ] ] separateby string_list ]
    [ [ desired [ pref_order ] ] strong separateby string_list ]
    [ [ desired [ pref_order ] ] affinity string_list ]
  } ] ]
} ] ] |

[ [ desired [ pref_order ] ] stripe from:int - to:int [ {
  [ [ desired [ pref_order ] ] confineto confineto_expr ]
  [ [ desired [ pref_order ] ] exclude exclude_expr ]
  [ [ desired [ pref_order ] ] select select_expr ]
  [ [ desired [ pref_order ] ] separateby string_list ]
  [ [ desired [ pref_order ] ] strong separateby string_list ]
  [ [ desired [ pref_order ] ] affinity string_list ]
  [ [ desired [ pref_order ] ] multipath multipath_expr ]
  [ [ desired [ pref_order ] ] mirror int [ {
    [ [ desired [ pref_order ] ] confineto confineto_expr ]
    [ [ desired [ pref_order ] ] exclude exclude_expr ]
    [ [ desired [ pref_order ] ] select select_expr ]
    [ [ desired [ pref_order ] ] separateby string_list ]
    [ [ desired [ pref_order ] ] strong separateby string_list ]
    [ [ desired [ pref_order ] ] affinity string_list ]
    [ [ desired [ pref_order ] ] multipath multipath_expr ]
  } ] ]
    [ [ desired [ pref_order ] ] mirror [ {
```

```
    [ [ desired [ pref_order ] ] confineto confineto_expr ]
    [ [ desired [ pref_order ] ] exclude exclude_expr ]
    [ [ desired [ pref_order ] ] select select_expr ]
    [ [ desired [ pref_order ] ] separateby string_list ]
    [ [ desired [ pref_order ] ] strong separateby string_list ]
    [ [ desired [ pref_order ] ] affinity string_list ]
    [ [ desired [ pref_order ] ] multipath multipath_expr ]
  } ] ]
} ] ] |

[ [ desired [ pref_order ] ] mirror_group mg_name:string [ {
  [ [ desired [ pref_order ] ] confineto confineto_expr ]
  [ [ desired [ pref_order ] ] exclude exclude_expr ]
  [ [ desired [ pref_order ] ] select select_expr ]
  [ [ desired [ pref_order ] ] separateby string_list ]
  [ [ desired [ pref_order ] ] strong separateby string_list ]
  [ [ desired [ pref_order ] ] affinity string_list ]
  [ [ desired [ pref_order ] ] multipath multipath_expr ]
  [ [ desired [ pref_order ] ] mirror int [ {
    [ [ desired [ pref_order ] ] confineto confineto_expr ]
    [ [ desired [ pref_order ] ] exclude exclude_expr ]
    [ [ desired [ pref_order ] ] select select_expr ]
    [ [ desired [ pref_order ] ] separateby string_list ]
    [ [ desired [ pref_order ] ] strong separateby string_list ]
    [ [ desired [ pref_order ] ] affinity string_list ]
    [ [ desired [ pref_order ] ] multipath multipath_expr ]
    [ [ desired [ pref_order ] ] striped boolean ]
    [ [ desired [ pref_order ] ] stripe from:int - to:int [ {
      [ [ desired [ pref_order ] ] confineto confineto_expr ]
      [ [ desired [ pref_order ] ] exclude exclude_expr ]
      [ [ desired [ pref_order ] ] select select_expr ]
      [ [ desired [ pref_order ] ] separateby string_list ]
      [ [ desired [ pref_order ] ] strong separateby string_list
]
      [ [ desired [ pref_order ] ] affinity string_list ]
    } ] ]
  } ] ]
  [ [ desired [ pref_order ] ] mirror [ {
    [ provides string_list ]
    [ [ desired [ pref_order ] ] confineto confineto_expr ]
    [ [ desired [ pref_order ] ] exclude exclude_expr ]
    [ [ desired [ pref_order ] ] select select_expr ]
    [ [ desired [ pref_order ] ] separateby string_list ]
    [ [ desired [ pref_order ] ] strong separateby string_list ]
    [ [ desired [ pref_order ] ] affinity string_list ]
    [ [ desired [ pref_order ] ] multipath multipath_expr ]
    [ [ desired [ pref_order ] ] striped boolean ]
    [ [ desired [ pref_order ] ] stripe from:int - to:int [ {
      [ [ desired [ pref_order ] ] confineto confineto_expr ]
      [ [ desired [ pref_order ] ] exclude exclude_expr ]
      [ [ desired [ pref_order ] ] select select_expr ]
      [ [ desired [ pref_order ] ] separateby string_list ]
```

```
      [ [ desired [ pref_order ] ] strong separateby string_list
]
      [ [ desired [ pref_order ] ] affinity string_list ]
    } ] ]
  } ] ]
} ] ] |
[ [ desired [ pref_order ] ] stripe all [ {
  [ [ desired [ pref_order ] ] confineto confineto_expr ]
  [ [ desired [ pref_order ] ] exclude exclude_expr ]
  [ [ desired [ pref_order ] ] select select_expr ]
  [ [ desired [ pref_order ] ] separateby string_list ]
  [ [ desired [ pref_order ] ] strong separateby string_list ]
  [ [ desired [ pref_order ] ] affinity string_list ]
  [ [ desired [ pref_order ] ] multipath multipath_expr ]
} ] ] |

[ [ desired [ pref_order ] ] log int [ {
  [ type dcm |dco | drl | raid5 ]
  [ [ desired [ pref_order ] ] confineto confineto_expr ]
  [ [ desired [ pref_order ] ] exclude exclude_expr ]
  [ [ desired [ pref_order ] ] select select_expr ]
  [ [ desired [ pref_order ] ] separateby string_list ]
  [ [ desired [ pref_order ] ] strong separateby string_list ]
  [ [ desired [ pref_order ] ] affinity string_list ]
  [ [ desired [ pref_order ] ] multipath multipath_expr ]
  [ [ desired [ pref_order ] ] striped boolean ]
  [ [ desired [ pref_order ] ] stripe from:int - to:int [ {
    [ [ desired [ pref_order ] ] confineto confineto_expr ]
    [ [ desired [ pref_order ] ] exclude exclude_expr ]
    [ [ desired [ pref_order ] ] select select_expr ]
    [ [ desired [ pref_order ] ] separateby string_list ]
    [ [ desired [ pref_order ] ] strong separateby string_list ]
    [ [ desired [ pref_order ] ] affinity string_list ]
    [ [ desired [ pref_order ] ] multipath multipath_expr ]
  } ] ]
} ] ] |
[ [ desired [ pref_order ] ] log all {
  type dcm |dco | drl | raid5
  [ [ desired [ pref_order ] ] confineto confineto_expr ]
  [ [ desired [ pref_order ] ] exclude exclude_expr ]
  [ [ desired [ pref_order ] ] select select_expr ]
  [ [ desired [ pref_order ] ] separateby string_list ]
  [ [ desired [ pref_order ] ] strong separateby string_list ]
  [ [ desired [ pref_order ] ] affinity string_list ]
  [ [ desired [ pref_order ] ] multipath multipath_expr ]
  [ [ desired [ pref_order ] ] striped boolean ]
  [ [ desired [ pref_order ] ] stripe from:int - to:int [ {
    [ [ desired [ pref_order ] ] confineto confineto_expr ]
    [ [ desired [ pref_order ] ] exclude exclude_expr ]
    [ [ desired [ pref_order ] ] select select_expr ]
    [ [ desired [ pref_order ] ] separateby string_list ]
    [ [ desired [ pref_order ] ] strong separateby string_list ]
```

```
    [ [ desired [ pref_order ] ] affinity string_list ]
    [ [ desired [ pref_order ] ] multipath multipath_expr ]
 } ] ]
 [ [ desired [ pref_order ] ] stripe all [ {
   [ [ desired [ pref_order ] ] confineto confineto_expr ]
   [ [ desired [ pref_order ] ] exclude exclude_expr ]
   [ [ desired [ pref_order ] ] select select_expr ]
   [ [ desired [ pref_order ] ] separateby string_list ]
   [ [ desired [ pref_order ] ] strong separateby string_list ]
   [ [ desired [ pref_order ] ] affinity string_list ]
   [ [ desired [ pref_order ] ] multipath multipath_expr ]
 } ] ]
} ] ]
```

confineto_expr

```
name:string |
name:string operator value:string |
name:quoted_string |
name:quoted_string operator value:quoted_string |
eachof(confineto_expr) |
anyof(confineto_expr) |
allof(confineto_expr) |
oneof(confineto_expr) |
noneof(confineto_expr) |
confineto_expr, confineto_expr
```

exclude_expr

```
name:string operator value:string |
name:quoted_string operator value:quoted_string |
allof(exclude_expr) |
exclude_expr, exclude_expr
```

multipath_expr

```
from:int [ - to:int ] name:string |
from:int [ - to:int ] name:quoted_string |
multipath_expr, multipath_expr
```

select_expr

```
confineto_expr
```

operator

```
<= | < | = | > | >= | !=
```

pref_order

int

User template

```
user_template ut_name:string [ {
  [ description quoted_string ]
  [ descriptionid descriptionid ]
  [ capability capabilities_expr ]
  [ rules [ {
    [ rules ]
  } ] ]
} ] ;
```

See “Format of user templates” on page 131.

capabilities_expr

```
c_name:string [ {
  parameter_list
} ]
| capabilities_expr, capabilities_expr
```

parameter_list

```
[ const ] attribute:string = value_expr
| parameter_list, parameter_list
```

value_expr

```
string | int
```

Storage pool

```
storage_pool sp_name:string [ {
  [ volume_templates vt_name_list ]
  [ description quoted_string ]
  [ descriptionid descriptionid ]
  [ autogrow boolean ]
  [ selfsufficient boolean ]
} ] ;
```

Template set

```
template_set sp_name:string [ {
  [ volume_templates vt_name_list ]
  [ description quoted_string ]
  [ descriptionid descriptionid ]
} ] ;
```

Storage pool set

```
storage_pool_set sp_name:string [ {
  [ storage_pools st_pool_list ]
  [ description quoted_string ]
  [ descriptionid descriptionid ]
} ] ;
```

st_pool_list

```
sp_name:string | st_pool_list, st_pool_list
```

Volume group

vg_rules

```
[ vg_rule ] | vg_rules vg_rule
```

vg_rule

```
[[ desired [pref_order]] confineto confineto_expr] |
[[ desired [pref_order]] exclude exclude_expr] |
[[ desired [pref_order]] select select_expr] |
[[ desired [pref_order]] separateby string_list] |
[[ desired [pref_order]] strong separateby string_list] |
[[ desired [pref_order]] affinity string_list] |
[[ desired [pref_order]] multipath multipath_expr]
```

confineto_expr

```
quoted_string |
name:quoted_string operator value:quoted_string |
eachof(confineto_expr) |
anyof(confineto_expr) |
oneof(confineto_expr) |
noneof(confineto_expr) |
allof(confineto_expr) |
confineto_expr , confineto_expr
```

exclude_expr

```
name:quoted_string operator value:quoted_string |
allof(exclude_expr) |
eachof(exclude_expr) |
exclude_expr, exclude_expr
```

select_expr

```
quoted_string |
name:quoted_string operator value:quoted_string |
eachof(select_expr) |
allof(select_expr) |
noneof(select_expr) |
anyof(select_expr) |
oneof(select_expr) |
select_expr, select_expr
```

multipath_expr

```
from:int [ - to: int ] name:quoted_string |
from:int [ - to: int ] name:string |
multipath_expr, multipath_expr
```

operator

```
<= | < | = | > | >= | !=
```

pref_order

```
1 <= pref_order <= maxint
```

Appendix B

Changing the allocation behavior of ISP

The behavior that ISP exhibits when allocating storage can be controlled by setting the value of the `allocation_priority_order` attribute. For example, this attribute determines if growing the storage pool is preferred to dropping the desired rules when there is insufficient storage to create a volume. The default behavior can be changed by editing the definition of the attribute in the `/etc/default/allocator` file, or, for a single invocation of the `vxassist` or `vxvoladm` command, by specifying the `-o allocation_priority=`*`value`* option as a command-line argument. The *`value`* argument can take an integer value from 1 to 12, as shown in the following table.

Value	Precedence order
1	S < T < A < D
2	S < T < D < A
3	S < A < T < D
4	S < D < T < A
5	S < A < D < T
6	S < D < A < T
7	A < S < T < D (default behavior)
8	D < S < T < A
9	A < S < D < T
10	D < S < A < T
11	A < D < S < T
12	D < A < S < T

In the table, the precedence order symbols have the following meaning:

Symbol	Description
`A`	Autogrow: If permitted by the value of the `AutoGrow` setting, the storage pool may be grown to increase the available storage.
`D`	Desired Rules: Rules may be dropped if insufficient storage is available.
`S`	Self-Sufficient: If permitted by the value of the `SelfSufficent` setting, templates may be added to the storage pool if insufficient storage is available.
`T`	Template Combination: Different template combinations may be used if insufficient storage is available.
`<`	Indicates the precedence in choosing an action when attempting to allocate storage. “X < Y” means that Y is chosen in preference to X. The rightmost items in the list are chosen before those to the left.

For example, the default behavior of ISP (equivalent to setting `allocation_policy_order=7`), A < S < T < D, means:

`T < D`	First, drop desired rules in preference to using a different template combination.
`S < T`	Secondly, use a different template combination in preference to adding more templates to the storage pool.
`A < S`	Finally, add more templates to the storage pool in preference to growing the storage pool.

If a set of capabilities are specified for a volume during creation, ISP may find that there are multiple sets of templates that provide the same set of capabilities. These sets are referred to as *template combinations*. Some of the available template combinations may provide additional capabilities (*extraneous capabilities*) to those that were requested for the volume. By preferring not to select a different template combination, you can avoid implicitly specifying the extraneous capabilities. This assumes that you would request these capabilities explicitly if you required them.

The templates that are associated with a storage pool define the characteristics of the volumes that can be created in that pool. The `selfsufficient` policy of the storage pool allows you to control whether ISP can redefine the pool definition automatically, and, if so, to what extent.

Appendix C

ISP configuration elements

This appendix lists and describes the following types of pre-defined elements that are available for use with Veritas Intelligent Storage Provisioning (ISP):

- Template set
- Volume template
- Capability
- Storage pool
- Storage pool set

These elements are defined in the `configuration_database.txt` file in the `/etc/vx/alloc` directory. Before modifying this file, make a backup copy so that you can reverse any changes that you make.

Template set

ConfineVolume

Ensures that a volume is confined to specified storage.

Provides capabilities: ConfineToSimilarStorage, ConfineToSpecificStorage.

Uses templates: ConfineToSimilarStorage, ConfineToSpecificStorage.

DataMirroring

Ensures that a volume has multiple copies of its data.

Provides capabilities: ArrayProductId, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, DataMirroring, DCOLogMirroring, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents.

Uses templates: ArrayProductId, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, DataMirroring, DCOLogMirroring, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents.

DataMirroringPrefabricatedRaid5

Ensures that a volume has multiple copies of data on prefabricated RAID-5 disks that are exported by an array.

Provides capabilities: ArrayProductId, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, DataMirroring, DCOLogMirroring, DCOLogStriping, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents, PrefabricatedRaid5.

Uses templates: ArrayProductId, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, DataMirroring, DCOLogMirroring, DCOLogStriping, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents, PrefabricatedRaid5.

DataMirroringPrefabricatedStriping

Ensures that a volume has multiple copies of data on prefabricated striped disks that are exported by an array.

Provides capabilities: ArrayProductId, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, DataMirroring, DCOLogMirroring, DCOLogStriping, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents, PrefabricatedStriping.

Uses templates: ArrayProductId, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, DataMirroring, DCOLogMirroring, DCOLogStriping, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents, PrefabricatedStriping.

DataMirrorStripe

Ensures that I/O from and to a volume is spread across multiple columns within mirrors.

Provides capabilities: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, DataMirrorStripe, DCOLogMirroring, DCOLogStriping, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents.

Uses templates: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, DataMirrorStripe, DCOLogMirroring, DCOLogStriping, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents.

DataStripeMirror

Ensures that I/O from and to a volume is spread across multiple columns, where each column has multiple copies of the data.

Provides capabilities: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, DataStripeMirror, DCOLogMirroring, DCOLogStriping, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents.

Uses templates: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, DataStripeMirror, DCOLogMirroring, DCOLogStriping, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents.

InstantSnapshottable

Ensures that a volume supports dirty region logging (DRL) and instant snapshots.

Provides capabilities: ConfineLogsToSimilarStorage, DCOLogMirroring, DCOLogStriping, InstantSnapshottable, LogsOnSeparateComponents.

Uses templates: ConfineLogsToSimilarStorage, DCOLogMirroring, DCOLogStriping, InstantSnapshottable, LogsOnSeparateComponents.

MultipathingThroughMirroring

Ensures that a volume can withstand the failure of a number of paths. The I/O from and to a volume can potentially be spread across all the paths.

Provides capability: DataMirroring, Multipathing.

Uses template: MultipathingThroughMirroring.

MultipathingThroughMultiplePaths

Ensures that a volume can withstand the failure of a specified number of paths. The I/O from and to the volume can potentially be spread across all these paths.

Provides capability: MultipathingThroughMultiplePaths.

Uses template: MultipathingThroughMultiplePaths.

PrefabricatedDataMirroring

Ensures that a volume uses prefabricated data mirroring configured within the disks exported by an array.

Provides capabilities: ArrayProductId, PrefabricatedDataMirroring.

Uses templates: ArrayProductId, PrefabricatedDataMirroring.

PrefabricatedRaid5

Ensures that a volume uses prefabricated RAID-5 disks that are exported by an array.

Provides capabilities: ArrayProductId, PrefabricatedRaid5.

Uses templates: ArrayProductId, PrefabricatedRaid5.

PrefabricatedStriping

Ensures that a volume uses prefabricated striped disks that are exported by an array.

Provides capabilities: ArrayProductId, PrefabricatedStriping.

Uses templates: ArrayProductId, PrefabricatedStriping.

Raid5Templates

Ensures that a volume uses parity to maintain redundant data.

Provides capabilities: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, ConfineLogsToSimilarStorage, LogsOnSeparateComponents, Raid5Capability, Raid5LogMirroring, Raid5LogStriping.

Uses templates: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, ConfineLogsToSimilarStorage, LogsOnSeparateComponents, Raid5LogStriping, Raid5Volume.

Striping

Ensures that I/O from and to a volume is spread across multiple columns.

Provides capabilities: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, Striping.

Uses templates: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, Striping.

StripingPrefabricatedDataMirroring

Ensures that I/O from and to a volume are spread across multiple columns, which are configured on prefabricated data mirror disks that are exported by an array.

Provides capabilities: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, PrefabricatedDataMirroring, Striping.

Uses templates: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, PrefabricatedDataMirroring, Striping.

Volume template

ArrayProductId

Provides capability: ArrayProductId.

ColumnsOnSeparateComponents

Provides capability: ColumnsOnSeparateComponents.

ConcatVolumes

Provides capability: ConcatVolumes.

ConfineColumnsToSimilarStorage

Provides capability: ConfineColumnsToSimilarStorage.

ConfineLogsToSimilarStorage

Provides capability: ConfineLogsToSimilarStorage.

ConfineMirrorsToSimilarStorage

Provides capability: ConfineMirrorsToSimilarStorage.

ConfineToSimilarStorage

Provides capability: ConfineToSimilarStorage.

ConfineToSpecificStorage

Provides capability: ConfineToSpecificStorage.

DataMirroring

Provides capability: DataMirroring.

Variable: `nmirs` Number of mirrors.

DataMirrorStripe

Provides capability: DataMirrorStripe.

Variables:

`ncols`	Minimum number of columns.
`nmaxcols`	Maximum number of columns.
`nmirs`	Number of mirrors.

DataStripeMirror

Provides capability: DataStripeMirror.

Variables:

`ncols`	Minimum number of columns.
`nmaxcols`	Maximum number of columns.
`nmirs`	Number of mirrors.

DCOLogMirroring

Provides capability: DCOLogMirroring.

DCOLogStriping

Provides capability: DCOLogStriping.

ExcludeSpecificStorage

Provides capability: ExcludeSpecificStorage.

InstantSnapshottable

Provides capability: InstantSnapshottable.

LogsOnSeparateComponents

Provides capability: LogsOnSeparateComponents.

MirrorsOnSeparateComponents

Provides capability: MirrorsOnSeparateComponents.

MultipathingThroughMirroring

Provides capabilities: DataMirroring, Multipathing.

Variable:

`nmirs`	Number of mirrors.

MultipathingThroughMultiplePaths

Provides capability: MultipathingThroughMultiplePaths.

PrefabricatedDataMirroring

Provides capability: PrefabricatedDataMirroring.

PrefabricatedRaid5

Provides capability: PrefabricatedRaid5.

PrefabricatedStriping

Provides capability: PrefabricatedStriping.

Raid5LogStriping

Provides capability: Raid5LogStriping.

Raid5Volume

Provides capabilities: Raid5Capability, Raid5LogMirroring.

Variables:		
	`ncols`	Minimum number of columns.
	`nlogs`	Number of logs.
	`nmaxcols`	Maximum number of columns.

Striping

Provides capability: Striping.

Variables:		
	`ncols`	Minimum number of columns.
	`nmaxcols`	Maximum number of columns.

Capability

ArrayProductId

A volume uses storage of the same type (`productid`).

Provided by volume template: ArrayProductId.

ColumnsOnSeparateComponents

The columns of a volume are separated at the specified component level, such as `"Controller"` or `"Enclosure"`.

Provided by volume template: ColumnsOnSeparateComponents.

Variable: `component` Name of component (default is `"Controller"`).

ConcatVolumes

The volume is concatenated.

Provided by volume template: ConcatVolumes.

ConfineColumnsToSimilarStorage

Each column uses only storage that has the same value for a specified attribute.

Provided by volume template: ConfineColumnsToSimilarStorage.

Variable: `name` Name of storage attribute (default is `"Enclosure"`).

ConfineLogsToSimilarStorage

Each log uses only storage that has the same value for a specified attribute.

Provided by volume template: ConfineLogsToSimilarStorage.

Variable: `name` Name of storage attribute (default is `"Enclosure"`).

ConfineMirrorsToSimilarStorage

Each mirror uses only storage that has the same value for a specified attribute.

Provided by volume template: ConfineMirrorsToSimilarStorage.

Variable: `name` Name of storage attribute (default is `"Enclosure"`).

ConfineToSimilarStorage

A volume uses only storage that has the same value for a specified attribute.

Provided by volume template: ConfineToSimilarStorage.

Variable: `name` Name of storage attribute (default is `VendorName`).

ConfineToSpecificStorage

A volume uses only storage that has the specified value for a specified attribute name.

Provided by volume template: ConfineToSpecificStorage.

Variables: `name` Name of storage attribute (no default).

`value` Value of storage attribute (no default).

DataMirroring

A mirrored volume maintains multiple copies of its data. This capability extends the DataRedundancy capability.

Provided by volume template: DataMirroring.

Variable: `nmirs` Number of mirrors (default is `2`).

DataMirrorStripe

A mirrored-stripe volume distributes I/O across multiple columns within mirrors. This capability extends the DataMirroring and Striping capabilities.

Provided by volume template: DataMirrorStripe.

Variables: `ncols` Minimum number of columns (default is `8`).

`nmaxcols` Maximum number of columns (default is `20`).

`nmirs` Number of mirrors (default is `2`).

DataRedundancy

A volume that maintains redundant data.

Extended by capability: Raid5Capability.

Extended by capability: DataMirroring.

DataStripeMirror

A striped-mirror volume distributes I/O across multiple columns, where each column has multiple copies of data. This capability extends the DataMirroring and Striping capabilities.

Provided by volume template: DataStripeMirror.

Variables:		
	`ncols`	Minimum number of columns (default is `8`).
	`nmaxcols`	Maximum number of columns (default is `20`).
	`nmirs`	Number of mirrors (default is `2`).

DCOLogMirroring

The number of DCO plexes (copies) to configure for a DCO volume.

Provided by volume template: DCOLogMirroring.

Variable:		
	`nlogs`	Number of DCO plexes (default is `1`).

DCOLogStriping

The number of columns to configure for a striped DCO volume.

Provided by volume template: DCOLogStriping.

Variables:		
	`ndcocols`	Minimum number of columns (default is `4`).
	`nmaxdcocols`	Maximum number of columns (default is `10`).

ExcludeSpecificStorage

A volume does not use storage that has the specified value for a specified attribute name.

Provided by volume template: ExcludeSpecificStorage.

Variables:		
	`name`	Name of storage attribute (no default).
	`value`	Value of storage attribute (no default).

InstantSnapshottable

A volume supports instant snapshots (full-sized or space-optimized). This capability extends the Snapshottable capability.

Provided by volume template: InstantSnapshottable.

LogsOnSeparateComponents

The logs of a volume are separated at the specified component level, such as `"Controller"` or `"Enclosure"`.

Provided by volume template: LogsOnSeparateComponents.

Variable: `component` Name of component (default is `"Enclosure"`).

MirrorsOnSeparateComponents

The mirrors of a volume are separated at the specified component level, such as `"Controller"` or `"Enclosure"`.

Provided by volume template: MirrorsOnSeparateComponents.

Variable: `component` Name of component (default is `"Enclosure"`).

Multipathing

Multipathing allows a volume to withstand failure of a number of paths to the disks. The I/O from and to the volume can potentially be spread across all the paths.

Extended by capability: MultipathingThroughMultiplePaths.

MultipathingThroughMultiplePaths

Multipathing allows a volume to withstand failure of the specified number of paths to the disks. The I/O from and to the volume can potentially be spread across all the paths. This capability extends the Multipathing capability.

Provided by volume template: MultipathingThroughMultiplePaths.

Variable: `npaths` Number of paths that are allowed to fail (default is `2`).

PrefabricatedDataMirroring

A volume uses prefabricated data mirroring that is configured on disks that are exported by an array. This capability extends the PrefabricatedDataRedundancy capability.

Provided by volume template: PrefabricatedDataMirroring.

Variable: `nmirs` Number of prefabricated mirrors to use (default is `2`).

PrefabricatedDataRedundancy

A volume uses prefabricated redundant disks that are exported by an array.

Extended by capabilities: PrefabricatedDataMirroring, PrefabricatedRaid5.

PrefabricatedRaid5

A volume uses prefabricated RAID-5 disks that are exported by an array. This capability extends the PrefabricatedDataRedundancy capability.

Provided by volume template: PrefabricatedRaid5.

PrefabricatedStriping

A volume uses prefabricated striped disks that are exported by an array.

Provided by volume template: PrefabricatedStriping.

Raid5Capability

A RAID-5 volume uses parity to implement data redundancy. This capability extends the DataRedundancy capability.

Provided by volume template: Raid5Volume.

Variable: `ncols` Minimum number of data columns (default is `8`).

`nmaxcols` Maximum number of data columns (default is `20`).

Note: An additional virtual column is always allocated to store parity information. This column is not included in the number of data columns that are specified.

Raid5LogMirroring

The number of RAID-5 log copies to configure for a RAID-5 volume.

Provided by volume template: Raid5Volume.

Variable:nlogsNumber of RAID-5 log copies (default value is `1`).

Raid5LogStriping

The number of columns to configure for striped RAID-5 logs.

Provided by volume template: Raid5LogStriping.

Variables: `nraid5cols` Minimum number of columns (default is `4`).

`nmaxraid5cols` Maximum number of columns (default is `10`).

Snapshottable

Volume snapshots of a volume may be taken.

Extended by capability: InstantSnapshottable.

Striping

A striped volume distributes I/O across multiple columns.

Provided by volume template: Striping.

Variables:

`ncols`	Minimum number of columns (default is `8`).
`nmaxcols`	Maximum number of columns (default is `20`).

Storage pool

any_volume_type

Supports volumes using any template from the configuration database.

Default policies: `autogrow=diskgroup`

`selfsufficient=host`

Associates volume templates: none.

mirror_stripe_volumes

Supports distribution of I/O from and to volumes across multiple columns within mirrors.

Default policies: `autogrow=diskgroup`

`selfsufficient=host`

Associates volume templates: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, ConfineToSimilarStorage, ConfineToSpecificStorage, DataMirrorStripe, DCOLogMirroring, DCOLogStriping, ExcludeSpecificStorage, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents, MultipathingThroughMultiplePaths.

mirrored_prefab_raid5_volumes

Supports use by volumes of multiple copies of data configured on prefabricated Raid-5 disks that are exported by an array.

Default policies: `autogrow=diskgroup`

`selfsufficient=host`

Associates volume templates: ArrayProductId, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, ConfineToSimilarStorage, ConfineToSpecificStorage, DataMirroring, DCOLogMirroring, DCOLogStriping, ExcludeSpecificStorage, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents, MultipathingThroughMultiplePaths, PrefabricatedRaid5.

mirrored_prefab_striped_volumes

Supports use by volumes of multiple copies of data configured on prefabricated striped disks that are exported by an array.

Default policies: `autogrow=diskgroup`

`selfsufficient=host`

Associates volume templates: ArrayProductId, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, ConfineToSimilarStorage, ConfineToSpecificStorage, DataMirroring, DCOLogMirroring, DCOLogStriping, ExcludeSpecificStorage, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents, MultipathingThroughMultiplePaths, PrefabricatedStriping.

mirrored_volumes

Supports volume with multiple copies of data.

Default policies: `autogrow=diskgroup`

`selfsufficient=host`

Associates volume templates: ArrayProductId, ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, ConfineToSimilarStorage, ConfineToSpecificStorage, DataMirroring, DCOLogMirroring, DCOLogStriping, ExcludeSpecificStorage, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents, MultipathingThroughMultiplePaths.

prefab_mirrored_volumes

Supports use by volumes of prefabricated data mirrors that are exported by an array.

Default policies: `autogrow=diskgroup`

`selfsufficient=host`

Associates volume templates: ArrayProductId, ConfineToSimilarStorage, ConfineToSpecificStorage, DCOLogStriping, ExcludeSpecificStorage, InstantSnapshottable, MultipathingThroughMultiplePaths, PrefabricatedDataMirroring.

prefab_raid5_volumes

Supports use by volumes of prefabricated RAID-5 disks that are exported by an array.

Default policies: `autogrow=diskgroup`
`selfsufficient=host`

Associates volume templates: ArrayProductId, ConfineToSimilarStorage, ConfineToSpecificStorage, DCOLogStriping, ExcludeSpecificStorage, InstantSnapshottable, MultipathingThroughMultiplePaths, PrefabricatedRaid5.

prefab_striped_volumes

Supports use by volumes of prefabricated striped disks that are exported by an array.

Default policies: `autogrow=diskgroup`
`selfsufficient=host`

Associates volume templates: ArrayProductId, ConfineToSimilarStorage, ConfineToSpecificStorage, DCOLogStriping, ExcludeSpecificStorage, InstantSnapshottable, MultipathingThroughMultiplePaths, PrefabricatedStriping.

raid5_volumes

Supports volumes which use parity to maintain redundant data.

Default policies: `autogrow=diskgroup`
`selfsufficient=host`

Associates volume templates: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, ConfineLogsToSimilarStorage, ConfineToSimilarStorage, ConfineToSpecificStorage, DCOLogStriping, ExcludeSpecificStorage, InstantSnapshottable, LogsOnSeparateComponents, Raid5Volume, MultipathingThroughMultiplePaths.

stripe_mirror_volumes

Supports distribution of I/O from and to volumes across multiple columns, where each column has multiple copies of data.

Default policies: `autogrow=diskgroup`
`selfsufficient=host`

Associates volume templates: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage,

ConfineLogsToSimilarStorage, ConfineMirrorsToSimilarStorage, ConfineToSimilarStorage, ConfineToSpecificStorage, DataStripeMirror, DCOLogMirroring, DCOLogStriping, ExcludeSpecificStorage, InstantSnapshottable, LogsOnSeparateComponents, MirrorsOnSeparateComponents, MultipathingThroughMultiplePaths.

striped_prefab_mirrored_volumes

Supports distribution of I/O from and to volumes across multiple columns that are configured on prefabricated data mirrors exported by an array.

Default policies: `autogrow=diskgroup`

`selfsufficient=host`

Associates volume templates: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, ConfineToSimilarStorage, ConfineToSpecificStorage, DCOLogStriping, ExcludeSpecificStorage, InstantSnapshottable, MultipathingThroughMultiplePaths, PrefabricatedDataMirroring, Striping.

striped_volumes

Supports distribution of I/O from and to volumes across multiple columns.

Default policies: `autogrow=diskgroup`

`selfsufficient=host`

Associates volume templates: ArrayProductId, ColumnsOnSeparateComponents, ConfineColumnsToSimilarStorage, ConfineToSimilarStorage, ConfineToSpecificStorage, DCOLogStriping, ExcludeSpecificStorage, InstantSnapshottable, MultipathingThroughMultiplePaths, Striping.

Storage pool set

mirrored_data_striped_clones

Supports data volumes with multiple copies of data, and snapshot volumes with I/O distributed across multiple columns.

Data storage pool type: mirrored_volumes.

Clone storage pool type: striped_volumes.

mirrored_prefab_raid5_data_mirrored_clones

Supports data volumes with multiple copies of data created from RAID-5 storage, and snapshot volumes with multiple copies of data.

Data storage pool type: mirrored_prefab_raid5_volumes.

Clone storage pool type: mirrored_volumes.

mirrored_prefab_stripe_data_striped_clones

Supports data volumes with multiple copies of data configured on prefabricated striped storage, and snapshot volumes with I/O distributed across multiple columns.

Data storage pool type: mirrored_prefab_striped_volumes.

Clone storage pool type: striped_volumes.

prefab_mirrored_data_prefab_striped_clones

Supports data volumes with multiple copies of data configured on prefabricated striped storage, and snapshot volumes with I/O distributed across multiple columns also configured on prefabricated striped storage.

Data storage pool type: prefab_mirrored_volumes.

Clone storage pool type: prefab_striped_volumes.

stripe_mirrored_data_striped_clones

Supports I/O from and to data volumes that are distributed across multiple columns, where each column has multiple copies of data, and snapshot volumes with I/O distributed across multiple columns.

Data storage pool type: stripe_mirror_volumes.

Clone storage pool type: striped_volumes.

striped_prefab_mirrored_data_striped_clones

Supports I/O from and to data volumes distributed across multiple columns constructed from mirrored storage, and snapshot volumes with I/O distributed across multiple columns.

Data storage pool type: striped_prefab_mirrored_volumes.

Clone storage pool type: striped_volumes.

Appendix D

Volume group definition syntax

This appendix describes the syntax of a `volumegroup` definition that may be used as input to the `vxassist` and `vxvoladm` commands. Volume groups are generally used to create several volumes at the same time within a storage pool and are given equal weighting by Veritas Intelligent Storage Provisioning (ISP) when allocating storage resources.

The syntax of a `volumegroup` definition is shown here:

```
volumegroup {
    [diskgroup "dgname"]
    [rules { volume-group-rules } ]
    volume "name" length {
        [cachevolume "cachevol"]
        [capability \
                'capability[(name=value,...)][,capability...]']
        [comment "comment"]
        [dcologlen length]
        [dm dm1,[dm2,...]]
        [drl on|true|yes|sequential|off|false|no]
        [excl on|off|true|false|yes|no]
        [fmr on|off|true|false|yes|no]
        [fstype type]
        [group "group"]
        [init active|none|zero]
        [iodelay delay]
        [iosize size]
        [layout "layout-type"]
        [max_ncolumn number]
        [max_nraid5column number]
        [min_ncolumn number]
        [min_nraid5column number]
        [mode mode]
        [nvol number]
        [nvols number]
        [pool "poolname"]
```

```
            [raid5_stripeunit width]
            [raid5loglen length]
            [regionsize size]
            [rules { volume-rules } ]
            [spare yes|no|only]
            [stripe_stripeunit width]
            [tag "volume_tag"]
            [tasktag "tag"]
            [type data|snapshot|cachevolume]
            [user "user"]
            [user_template "ut1"["ut2"...]]
            [usetype "volume-usage-type"]
            [volume_template "vt1"["vt2"...]]
            [vxvmtaskid taskid]
        }
        [volume "name" length { ...
        }]
    };
```

The `volumegroup` fields in this definition are:

`diskgroup` *dgname*	The name of the disk group in which the volumes are to be created.
`rules {` *volume-group-rules* `}`	The rules that are to be applied to all volumes in this volume group.
`volume "`*name*`"` *length*	The name of a volume to be created and its size.

The `volume` fields in the definition are:

`cachevolume "`*cachevol*`"`	The name of the cache volume on which a space-optimized snapshot volume is to be created.
`capability '`*capability*`[(`*name*`=`*value*`,...)][,`*capability*`...]'`	The capabilities that are desired for the volume.
`comment "`*comment*`"`	A description of the volume.
`dcologlen` *length*	The size of a DCO volume whose creation is implied by the specified capabilities or templates.
`dm` *dm1*`,[`*dm2*`,...]`	A list of disk media which can be allocated to a volume.
`drl on|true|yes|sequential|off|false|no`	Whether DRL, sequential DRL, or no DRL is enabled on a volume.

`excl on|off|true|false|yes|no`	Whether a volume in a cluster-shareable disk group can be opened by only one node at a time.
`fmr on|off|true|false|yes|no`	Whether FastResync is enabled on a volume.
`fstype` *`type`*	Specifies the file system type for a volume.
`group "`*`group`*`"`	The group ownership of a volume.
`init active|none|zero`	The type of initialization to perform on a volume.
`iodelay` *`delay`*	The delay in milliseconds between copy operations performed during recovery of a plex.
`iosize` *`size`*	The size of each region that is resynchronized during recovery of a plex.
`layout "`*`layout-type`*`"`	The plex layout that is to be created.
`max_ncolumn` *`number`*	The maximum number of columns in a striped volume.
`max_nraid5column` *`number`*	The maximum number of columns in a RAID-5 volume.
`min_ncolumn` *`number`*	The minimum number of columns in a striped volume.
`min_nraid5column` *`number`*	The minimum number of columns in a RAID-5 volume.
`mode` *`mode`*	The permissions to be applied to a volume, for example 644.
`nvol` *`number`*	The number of volumes to be created. The volumes are named using the specified volume name as a prefix with a number starting from 1 appended. If any volume names already exist in the same pool with the same prefix and an appended number, numbering starts at the highest number found plus 1.
`nvols` *`number`*	Alias for `nvol`.
`pool "`*`poolname`*`"`	The name of the storage pool.

`raid5_stripeunit` *`width`*	The stripe unit width for a RAID-5 column.
`raid5loglen` *`length`*	The size of a RAID-5 log whose creation is implied by the specified capabilities or templates.
`regionsize` *`size`*	The size of each region that is tracked by a bit in the maps within a version 20 DCO volume. The value of *size* must be a power of 2 and be greater than or equal to 16k (16KB). The default value is 64k (64KB).
`rules { volume-rules }`	The rules that are to be applied during volume creation.
`spare yes|no|only`	Whether spare disks can be used for hot-relocation, or only spare disks are to be used.
`stripe_stripeunit` *`width`*	The stripe unit width.
`tag "volume_tag"`	A tag that is assigned to the volume.
`tasktag "tag"`	A tag that can be used to track the volume creation task.
`type data|snapshot|cachevolume`	Whether a volume is to be used for user data (such as a database or file system), as a full-sized instant snapshot volume, or as a cache volume for space-optimized instant snapshots.
`user "user"`	The user ownership of a volume.
`user_template` *`ut1`*[,*`ut2`*...]	The user templates that are to be used to create a volume.
`usetype "volume-usage-type"`	The usage type of a volume, such as `fsgen` or `raid5`.
`volume_template` *`vt1`*[,*`vt2`*...]	The templates that are to be used to create a volume.
`vxvmtaskid` *`taskid`*	VxVM task ID.

Multiple volumes may be defined within a volume group, and multiple volume groups may be defined in a definitions file.

Appendix E

Equivalences between vxassist attributes and ISP rules

The following table shows the equivalent ISP rules for attributes in `vxassist` that specify volume configuration.

vxassist attribute	ISP rules	Description
layout=mirror-stripe	mirror nmir { stripe max_ncolumn-min_ncolumn }	Mirrored-stripe volume.
layout=stripe-mirror	stripe max_ncolumn-min_ncolumn { mirror nmir }	Striped-mirror volume.
layout=mirror,stripe	mirror nmir stripe max_ncolumn-min_ncolumn	Mirrored-stripe volume.
layout=stripe,mirror	mirror nmir stripe max_ncolumn-min_ncolumn	Mirrored-stripe volume.
layout=mirror	mirror nmir	Mirrored volume.
layout=raid5	stripe max_nraid5column-min_nraid5column parity true log 1 { type raid5 }	RAID-5 volume.
layout=stripe	stripe max_ncolumn-min_ncolumn	Striped volume.

vxassist attribute	ISP rules	Description
layout=mirror-concat	mirror nmir	Mirrored volume.
layout=concat-mirror	mirror nmir	Concatenated-mirror volume.
layout=log	log nlog { type dco }	DCO volume.
layout=raid5log	log nlog { type raid5 }	RAID-5 log volume
layout=regionlog	log nlog { type drl }	DRL log plex.
mirror=2	mirror 2	Number of mirrors.
mirror=enclosure	mirror nmir { separateby "enclosure" }	Mirror across enclosures.
mirror=2,enclosure	mirror 2 { separateby "enclosure" }	Two mirrors on separate enclosures.
stripe=ctlr	stripe ncols { separateby "ctlr" }	Stripes on separate controllers.
mirrorconfine=diskclass	mirror all { confineto "diskclass" }	Confine creation of mirrors to a given class of disks.
nmir=2	mirror nmir	Default mirrored volume (equivalent to layout=mirror,concat).
ncols=10	stripe ncols	Default striped volume (equivalent to layout= stripe).
nmir=2 ncol=10	mirror nmir stripe ncols	Default mirrored-stripe volume (equivalent to layout=mirror,stripe).
fmr=on	log nlog { type dco }	DCO volume (enables the use of FastResync).
fastresync=on	log nlog { type dco }	DCO volume (enables the use of FastResync).
ndcmlog=2	log 2 { type dcm }	DCM log with 2 mirrors.
ndcomirror=2	log 2 { type dco }	DCO volume with 2 mirrors.
nlog=2	log 2 { type dco }.	DCO volume with 2 mirrors.
nraid5log=2	log 2 { type raid5 }	RAID-5 log with 2 mirrors.

vxassist attribute	ISP rules	Description
nraid5stripe=10	stripe 10 parity true log 1 { type raid5 }	RAID-5 volume with 10 columns.
nregionlog=2	log 2 { type drl }	Two DRL log plexes.
Nstripe=10	stripe 10	10 stripes.
wantalloc=enclosure	desired confineto "enclosure"	Preferred storage allocation is confined to one enclosure.
wantmirror=enclosure	mirror all { desired confineto "enclosure" }	Preferred storage allocation is confined to one enclosure.
wantmirrorconfine=controller	mirror all { desired confineto "controller" }	Preferred storage allocation is confined to one controller.

Appendix F

Command summary

This appendix summarizes the usage and purpose of important commands for the Veritas Intelligent Storage Provisioning (ISP) feature of Veritas Volume Manager (VxVM).

References are included to longer descriptions in the remainder of this book. For detailed information about an individual command, refer to the appropriate manual page in the 1M section.

See “Examples of using ISP” on page 209.

Table F-1 Creating and administering storage pools

Command	Description

`vxpool [-g diskgroup] \`
`  create storage_pool \`
`  [dm=dm1[,dm2...] \`
`  [description="description"] \`
`  [autogrow=1|2] \`
`  [selfsufficient=1|2|3] \`
`  [rules=rule[,rule ...]`
	Creates a storage pool.
	See “Creating a storage pool” on page 52.

`vxpool [-g diskgroup] \`
`  adddisk storage_pool \`
`  dm=dm1[,dm2,...]`
	Adds one or more disks to a storage pool.
	See “Adding disks to a storage pool” on page 55.

`vxpool [-g diskgroup] \`
`  rmdisk storage_pool \`
`  dm=dm1[,dm2,...]`
	Removes one or more disks from a storage pool.
	See “Removing disks from a storage pool” on page 55.

`vxpool [-g diskgroup] \`
`  assoctemplate storage_pool \`
`  t1[,t2,...]`
	Associates one or more templates with a storage pool.
	See “Associating templates with a storage pool” on page 55.

Table F-1 Creating and administering storage pools

Command	Description
`vxpool [-g diskgroup] \ assoctemplateset\ storage_pool \ ts1[,ts2,...]`	Associates one or more template sets with a storage pool. See "Associating template sets with a storage pool" on page 56.
`vxpool [-g diskgroup] \ distemplate storage_pool \ t1[,t2,...]`	Dissociates one or more templates from a storage pool. See "Dissociating templates from a storage pool" on page 56.
`vxpool [-g diskgroup] \ print storage_pool \ [storage_pool ...]`	Displays information about storage pools. See "Displaying information about storage pools" on page 57.
`vxpool [-g diskgroup] \ getproperty storage_pool`	Displays policies for a storage pool. See "Displaying storage pool policies" on page 57.
`vxpool [-g diskgroup] \ setproperty storage_pool \ [autogrow={1|pool}|\ {2|diskgroup}] [selfsufficient={1|pool}|\ {2|diskgroup}|{3|host}]`	Sets policies on a storage pool. See "Changing the policies on a storage pool" on page 57.
`vxpool [-g diskgroup] list`	Lists storage pools in a disk group. See "Listing storage pools within a disk group" on page 58.
`vxpool [-g diskgroup] \ rename storage_pool new_name`	Renames a storage pool. See "Renaming a storage pool" on page 58.
`vxpool [-g diskgroup] [-r] \ delete storage_pool`	Deletes a storage pool. See "Deleting a storage pool" on page 59.

Table F-2 Creating and administering application volumes

Command	Description
`vxassist [-g diskgroup] [-b] \ maxsize [storage_specification] \ [attribute=value ...]`	Determines the maximum volume size. See "Determining the maximum volume size" on page 63.
`vxassist [-g diskgroup] \ [-P storage_pool] [-b] \ make volume length \ [storage_specification] \ [attribute=value ...]`	Creates a volume. See "Creating volumes" on page 64.
`vxassist [-g diskgroup] \ [-P storage_pool] [-b] \ make volume length \ capability='DataMirroring' \ init=active`	Creates a volume with 2 mirrors and makes it immediately available for use. See "Creating a mirrored volume" on page 66.
`vxassist [-g diskgroup] \ [-P storage_pool] [-b] \ make volume length \ capability='Striping(ncols=N)'`	Creates a striped volume with *N* columns. See "Creating a striped volume" on page 67.
`vxassist [-g diskgroup] \ [-P storage_pool] [-b] \ make volume length \ capability='PrefabricatedRaid5'\ rules='confineto \ "VendorName"="Name"'`	Creates a RAID-5 volume on a device that is exported by an array made by a specified vendor. See "Creating a RAID-5 volume on prefabricated storage" on page 68.
`vxassist -M make < filename`	Creates multiple volumes from a specification file. See "Creating multiple volumes as a volume group" on page 71.
`vxassist [-g diskgroup] maxgrow\ volume [storage_specification] \ [attribute=value ...]`	Determines the maximum growable size of a volume. See "Determining the maximum size of a volume" on page 77.
`vxassist [-g diskgroup] \ {growby | growto}\ volume length \ [storage_specification] \ [attribute=value ...]`	Grows a volume. See "Increasing the size of a volume to a specified length" on page 77. See "Increasing the size of a volume by a specified amount" on page 77.

Table F-2 Creating and administering application volumes

Command	Description
`vxassist [-g diskgroup] \ {shrinkby | shrinkto}\ volume length \ [storage_specification]\ [attribute=value ...]`	Shrinks a volume. See "Reducing the size of a volume to a specified length" on page 78. See "Reducing the size of a volume by a specified amount" on page 78.
`vxassist [-g diskgroup] \ evacuate disk volume length \ !dm:dmname ...`	Evacuates a volume from one or more specified disks. See "Evacuating a volume" on page 81.
`vxassist [-g diskgroup] remove \ volume volume`	Deletes a volume. See "Removing a volume" on page 82.
`vxassist [-g diskgroup] verify \ volume`	Verifies the intent of a volume. See "Verifying the intent of a volume" on page 89.

Appendix G

Examples of using ISP

This appendix lists examples of using the `vxassist` and other commands to administer the Veritas Intelligent Storage Provisioning (ISP) feature.

- Creating volumes
- Resizing a volume
- Adding or removing mirrors
- Adding or removing columns
- Adding or removing logs
- Evacuating data from a volume
- Segregating database components

See "Examples of using ISP from the command line" on page 45.

See the `vaxassist`(1M) manual page.

Creating volumes

Create a 10-gigabyte concatenated volume:

```
# vxassist -g exdg -P expool make vol1 10g
```

Create a 2-way mirrored volume using rules:

```
# vxassist -g exdg -P expool make vol1 10g rules='mirror 2'
```

Create a 7-column striped volume using rules:

```
# vxassist -g exdg -P expool make vol1 10g rules='stripe 7'
```

Create a 2-way mirrored volume using the `DataMirroring` capability:

```
# vxassist -g exdg -P expool make vol1 10g \
  capability='DataMirroring(nmirs=2)'
```

Create a 2-way mirrored volume with each mirror on a separate enclosure:

```
# vxassist -g exdg -P expool make vol1 10g \
  capability='DataMirroring(nmirs=2),\
  MirrorsOnSeparateComponents(component="Enclosure")'
```

Create a 2-way mirrored volume that is enabled for the creation of instant snapshots:

```
# vxassist -g exdg -P expool make vol1 10g \
  capability='DataMirroring(nmirs=2), InstantSnapshottable'
```

Create a volume using only disks from enclosure `e1`:

```
# vxassist -g exdg -P expool vol1 10g \
  rules='confineto "Enclosure"="e1"'
```

or alternatively:

```
# vxassist -g exdg -P expool make vol1 10g \
  capability='ConfineToSpecificStorage(name="Enclosure",\
  value="e1")'
```

Create a volume using disks on the same controller:

```
# vxassist -g exdg -P expool make vol1 10g \
  rules='confineto "Controller"'
```

or alternatively:

```
# vxassist -g exdg -P expool make vol1 10g \
  capability='ConfineToSimilarStorage (name="Controller")'
```

Create a RAID-5 volume with a 2-way mirrored RAID-5 log and a 2-way mirrored DCO volume:

```
# vxassist -g exdg -P expool make vol1 10g \
  capability='Raid5Capability,\
Raid5LogMirroring(nlogs=2), DCOLogMirroring(nlogs=2)'
```

Create a volume using a RAID-5 device that has been implemented in hardware (such as a RAID-5 LUN in an EMC Symmetrix array):

```
# vxassist -g exdg -P expool make vol1 10g \
  capability='PrefabricatedRaid5'
```

Create a striped volume with a minimum of 10 columns using mirrored LUNs from enclosure `e1`, and for which instant snapshots can be taken:

```
# vxassist -g exdg -P expool make vol1 10g \
  capability='PrefabricatedDataMirroring, Striping(ncols=10),\
InstantSnapshottable' rules='confineto "Enclosure"="e1"'
```

or alternatively:

```
# vxassist -g exdg -P expool make vol1 10g \
  capability='PrefabricatedDataMirroring, Striping(ncols=10),\
InstantSnapshottable,\
ConfineToSpecificStorage(name="Enclosure",value="e1")'
```

Create a 4-way mirrored volume in which each mirror uses disks from the same enclosure, and for which instant snapshots can be taken:

```
# vxassist -g exdg -P expool make vol1 10g \
  capability='DataMirroring(nmirs=2),InstantSnapshottable,\
ConfineMirrorsToSimilarStorage(name="Enclosure")'
```

Resizing a volume

Note: The following examples show how to resize volumes that do not contain file systems (volume type `gen`). If an application is using the volume to store data, use the appropriate utility to reduce the size of the data object before reducing the size of the volume, or to increase the size of the data object after increasing the size of the volume.

For volumes that contain file systems (volume type `fsgen`), use the `vxresize` command to change both the size of the file system and the volume.

Grow a volume by 5 gigabytes:

```
# vxassist -g exdg growby vol1 5g
```

Grow an existing 10-gigabyte volume to 15 gigabytes without using disks from the enclosure `e1`:

```
# vxassist -g exdg growto vol1 15g \
  use_storage='noneof("Enclosure"="e1")'
```

Shrink a volume by 10%:

```
# vxassist -g exdg growby vol1 10%
```

Shrink an existing 15-gigabyte volume to 10 gigabytes:

```
# vxassist -g exdg shrinkto vol1 10g
```

Adding or removing mirrors

Add a mirror to an existing volume:

```
# vxassist -g exdg mirror vol1
```

Add two mirrors to a volume:

```
# vxassist -g exdg mirror vol1 nmirror=2
```

Add a mirror to the volume `vol1`, where the new mirror is like the existing plex `p1`:

```
# vxassist -g exdg mirror vol1 plexname=p1
```

Remove a mirror from a volume:

```
# vxassist -g exdg remove mirror vol1
```

Remove the plex `p1` from the volume `v1`:

```
# vxassist -g exdg remove mirror vol1 plexnames=p1
```

Adding or removing columns

Add a column to a volume, and grow the volume while doing this:

```
# vaxassist -g exdg add column vol1 layout=grow
```

Add 4 columns to a volume, and grow the volume while doing this:

```
# vaxassist -g exdg add column vol1 ncols=4 layout=grow
```

Add a column, but do not grow the volume:

```
# vaxassist -g exdg add column vol1 layout=nogrow
```

Remove a column, and shrink the volume while doing this:

```
# vaxassist -g exdg remove column vol1 layout=shrink
```

Remove a column, but do not shrink the volume:

```
# vaxassist -g exdg remove column vol1 layout=noshrink
```

Adding or removing logs

Add a DCO log to a volume:

```
# vaxassist -g exdg addlog vol1 logtype=dco
```

Add a DCO log to volume `vol1`, and make the new log like the existing log `p1`:

```
# vaxassist -g exdg addlog vol1 lognames=p1 logtype=dco
```

Remove a RAID-5 log from a volume:

```
# vaxassist -g exdg remove log vol1 logtype=raid5
```

Remove the RAID-5 log named `p1` from a volume:

```
# vaxassist -g exdg remove log vol1 lognames=p1 logtype=raid5
```

Evacuating data from a volume

Evacuate subdisk `sd01` from the volume `vol1`:

```
# vaxassist -g exdg evacuate subdisk vol1 sd=sd01
```

Evacuate subdisks `sd11` and `sd12` from volume `vol1`:

```
# vaxassist -g exdg evacuate subdisk vol1 sd=sd11,sd12
```

Evacuate disks `disk01` and `disk01` from volume `vol1`:

```
# vaxassist -g exdg evacuate disk vol1 \!disk01 \!disk02
```

Evacuate columns 4 and 7 of volume `vol1`:

```
# vaxassist -g exdg evacuate column vol1 columnnames=4,7
```

Evacuate mirror `p1` of volume `vol1`:

```
# vaxassist -g exdg evacuate mirror vol1 plexnames=p1
```

Evacuate columns 3 and 4 of mirror `p1` from volume `vol1`:

```
# vaxassist -g exdg evacuate column vol1 columnnames=3,4 \
  plexnames=p1
```

Segregating database components

To achieve best performance, it is usual to place the components of a database, such as the redo log, table spaces and indexes, on different disks. This can be accomplished by using the disk tagging feature, which allows you to define new characteristics and then assign these to your disks.

Note: The disk tagging may be accessed by selecting a disk in the Veritas Enterprise Administrator, and then selecting **Actions > Annotate Disks**. This feature is also available from the command line by using the `vxdisk settag` command. See the `vxdisk`(1M) manual page for more information.

Disk tagging can be used in conjunction with volume rules to reserve disks for particular volumes. By tagging disks with a suitable attribute name and value such as `reserved_for` and `redo_log`, you can use a `confineto` rule to ensure that the volume for a redo log is only allocated space from those disks:

```
confineto "reserved_for"="redo_log"
```

Conversely when creating volumes that are not to be used for the redo logs, you can specify an `exclude` rule:

```
exclude "reserved_for"="redo_log"
```

This technique allows you to share appropriately tagged disks among a set of volumes of the same type. For example, if you specify the same confinement rule when creating several redo log volumes in a storage pool, those volumes share the disks that are reserved for that purpose.

Appendix H

Configuring ISP to work with SAL

The SAN Access Layer (SAL) provides information about storage, hosts, and connectivity between hosts, and storage, HBAs, volumes and file systems. You can configure ISP to understand and consume information that is provided by SAL. The storage attributes provided by SAL can then be used with ISP rules. This appendix describes how to configure ISP and the SAN Access Layer (SAL) so that they can work together.

Enabling ISP to work with SAL

To enable ISP to receive information from SAL

1. To allow ISP to contact SAL, add the following line to the `/etc/default/vxassist` file:

   ```
   salcontact=yes
   ```

   If this attribute is set to no or is not present, ISP cannot contact SAL.

2. Define the SAL Primary Host and SAL Primary Port by adding `sal_primary` and `sal_primary_port` definitions to the `/etc/default/vxassist_sal_info` file, as shown in the following sample entries:

   ```
   sal_primary=private.cosmos.com
   sal_primary_port=2802
   ```

   This example enables ISP to communicate with the SAL primary `private.cosmos.com` on port number 2802.

   If this information is not specified, the host running ISP is assumed to be the SAL primary, and the default port to be 2802.

3. To establish a communication channel successfully, ISP must authenticate itself with SAL using a user name and password. The user name is that of

the account under which the VEA service is running, usually `root`. Use the `vxsalcmd` command to add this user account with SAL administrator privileges, and set an appropriate password for it. To allow ISP to have transparent access to SAL, also use the `vxsalcmd` command to store the password on the local machine in encrypted form.
See "Configuring root as a SAL user" on page 216.
.

Note: The `vxsalcmd` command is provided by SAL to manage SAL user accounts. It is not a Veritas Volume Manager command.

4 Edit the `etc/default/vxassist` file, and add an entry to define the user name that ISP uses for authentication. In the following example, the root account is used:

```
sal_username=root
```

Configuring root as a SAL user

To add the root user on the remote host as a SAL user with SAL administrator privileges on the SAL primary

1 Create a file, such as `/tmp/add_user`, that contains the following single line definition of the new SAL user:

```
user add root@island.veritas.com clydenw Admin
```

This defines `root` on the remote host (`island.cosmos.com` in this example) as a SAL user with SAL administrator privileges, and with a password set to `clydenw`.

2 Use the following command to add the new SAL user on the SAL primary (`private.cosmos.com` in this example):

```
# vxsalcmd private.cosmos.com:2802 -u admin -f /tmp/add_user
```

You are prompted to enter the password for the SAL administrator, in this example named `admin`.

To view the new SAL user account, enter the following command:

```
# vxsalcmd private.cosmos.com:2802 -u admin user list
```

Again, you are prompted for the SAL administrator's password.

3 To allow ISP to communicate transparently with SAL, a local copy of the encrypted account information for the SAL user must be set up. Create a file, such as `/tmp/maintain_user`, that contains the following single line definition of the SAL user's password:

```
user rem clydenw
```

4 Use the following command to create the local SAL account file:

```
# vxsalcmd private.cosmos.com:2802 -f /tmp/maintain_user
```

To confirm that the password has been set up correctly, enter the following command:

```
# vxsalcmd private.cosmos.com:2802 user list
```

This command should now list all the configured users without prompting for the SAL administrator's password.

5 Remove the `/tmp/add_user` and `/tmp/maintain_user` files as these contain passwords in clear text.

Note: If the SAL user's password is not stored locally, ISP attempts to authenticate with SAL as the user `admin` with the password `passwd`.

If ISP fails to authenticate with SAL, it can neither see nor use any of the associated storage attributes.

Index

Symbols

A

D

E

F

G

H

I

L

M

N

O

P

R

S

V