VERITAS®

VERITAS® Cluster Server

Installation Guide

Version 1.1

Solaris

August 1999

P/N 100-001168

Table of Contents

Preface

This guide provides information on:

- How to install VERITAS® Cluster Server™ (VCS) on Solaris.
- The VCS processes, or “agents,” that work with VCS resources.

It is intended for system and network administrators responsible for installing and configuring VCS.

For more information on installing and configuring VCS, see the *VERITAS Cluster Server Quick-Start Guide, An Example with NFS*.

For more information on using VCS, see the *VERITAS Cluster Server User’s Guide*.

For more information on the API provided by the VCS agent framework, and for instructions on how to build and test an agent, see the *VERITAS Cluster Server Agent Developer’s Guide*.

How This Guide Is Organized

Chapter 1, “Installing VCS,” instructs you on how to install and configure your hardware, and how to install the VCS software.

Chapter 2, “VCS Resource Types and Agents,” explains how agents work, and provides a list of agents provided with VCS.

Chapter 3, “Troubleshooting VCS,” provides information on how to reconcile major and minor numbers.

Technical Support

For assistance with this product, contact VERITAS Customer Support:

U.S. and Canadian Customers: 1-800-342-0652

International Customers: +1 (650) 335-8555

Fax: (650) 335-8428

Email: support@veritas.com

Conventions

Typeface	**Usage**
`courier`	computer output, files, attribute names, device names, and directories
`courier (bold)`	user input and commands, keywords in grammar syntax
italic	new terms, titles, emphasis, variables replaced with a name or value
italic (bold)	variables within a command
Symbol	**Usage**
%	C shell prompt
$	Bourne/Korn shell prompt
#	Superuser prompt (for all shells)
\	Command-line continuation if last character in line. Not to be confused with an escape character.

Installing VCS

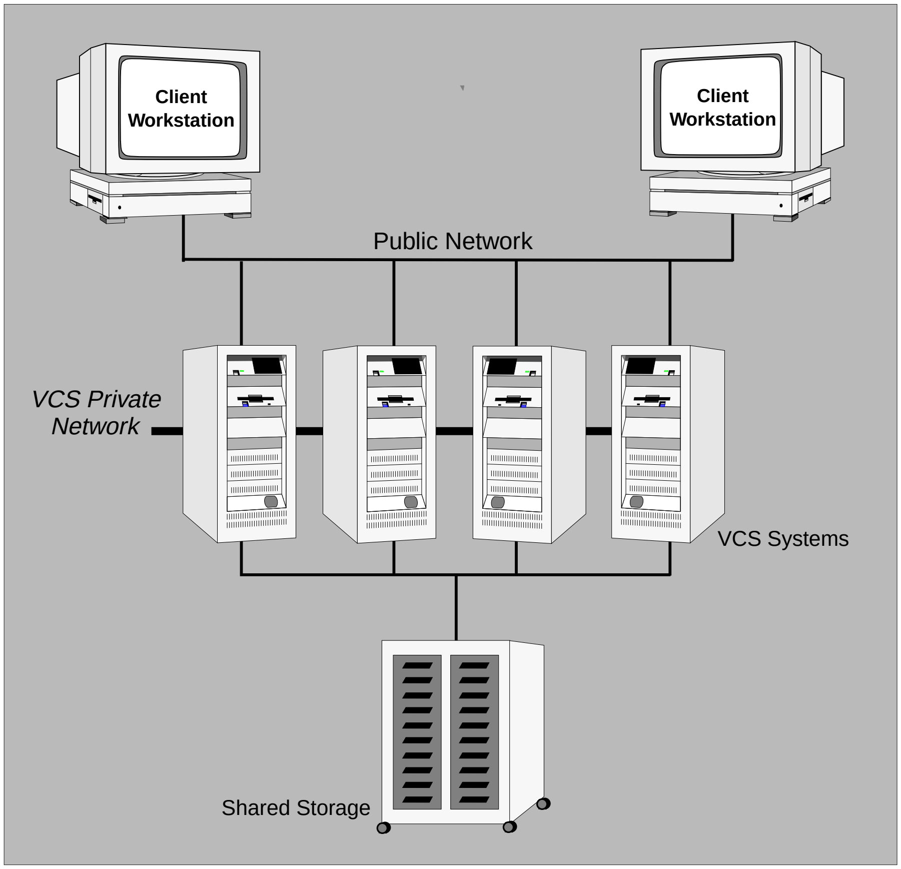
1

VERITAS® Cluster Server™ (VCS) is the latest high-availability solution for cluster configurations. VCS enables you to monitor systems and application services, and to restart services on a different system when hardware or software fails.

VCS Basics

A single VCS cluster consists of multiple systems connected in various combinations to shared storage devices. VCS monitors and controls applications running in the cluster, and restarts applications in response to a variety of hardware or software faults. Client applications continue operation with little or no downtime. In some cases, such as NFS, this continuation is transparent to high-level applications and to users. In other cases, the operation must be retried; for example, reloading a Web page.

The illustration on page 12 shows a typical VCS configuration of four systems connected to shared storage. Client workstations receive service over the public network from applications running on the VCS systems. VCS monitors the systems and their services, and the systems communicate over a private network.



Example of a Four-System VCS Cluster

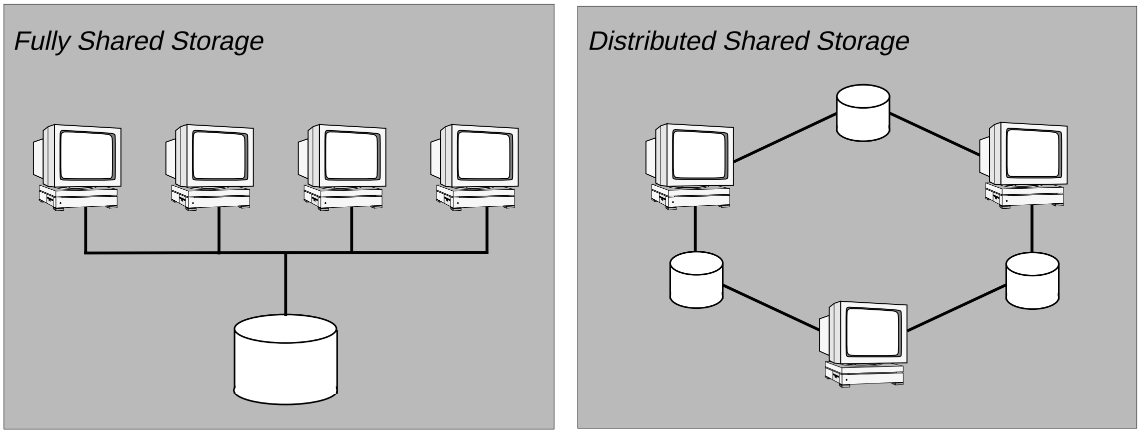
Multiple Systems

The VCS engine runs on each system in the cluster, and each system is a replicated state machine. State information is propagated between systems via a private network. This enables systems to share identical state information about all resources, and to recognize which systems are active members of the cluster, which are joining or leaving the cluster, and which have failed. Dual communication channels are required to guard against network partitions.

Shared Storage

A VCS hardware configuration consists of multiple systems that can be connected to shared storage via I/O channels. Shared storage allows multiple systems an access path to the same data, and enables VCS to restart applications on alternate systems when a system fails, thus ensuring high availability.

The figures below illustrate the flexibility of VCS shared storage configurations. (Note that VCS systems can only access storage that is physically attached.)



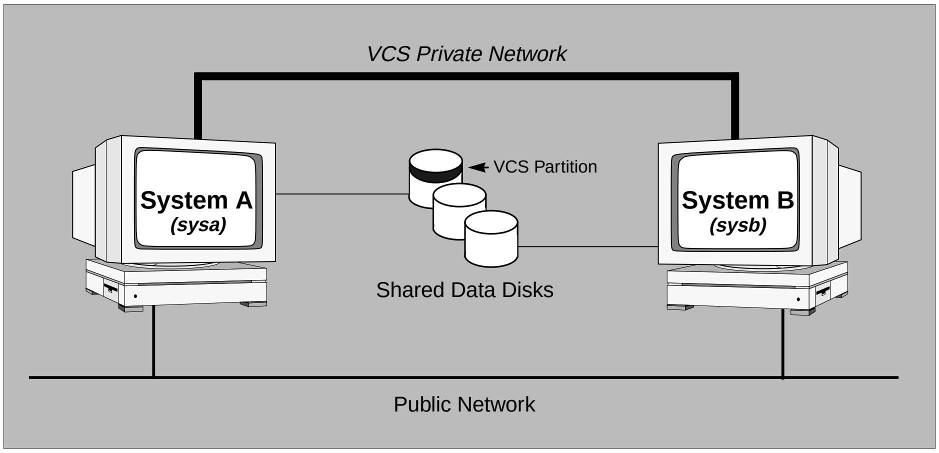
Two Examples of Shared Storage Configurations

Communication Channels

There are two types of channels available for VCS communication: networks and communication disks. (The term disk refers to a region of a physical disk partition.)

Each cluster configuration requires at least two communication channels, and one of them must be a network channel. The remaining channels may be a combination of networks and communication disks. This requirement protects your cluster against network partitioning. (For more information on network partitioning, refer to the *VERITAS Cluster Server User's Guide*.)

VCS supports a maximum of eight network channels and four communication disks. VCS communication disks must be connected to all systems in the cluster. We recommend configuring at least one disk on each I/O chain between the systems. (For configurations using communication disks, the cluster size is currently limited to eight systems.)



Two Systems Connected by a Single Ethernet and Communication Disk

Even with two communication channels, VCS can partition if all channels fail simultaneously; for example, if power fails to all private network hubs and you have not configured a communication disk. To protect application data in this scenario, VCS offers another type of disk monitoring facility called *service group heartbeat disks*. Heartbeat disks are associated with service groups, and must be accessible from each system that can run the service group. The number of systems that can be supported by this feature is unlimited.

The VCS installation procedures (page 16) allocate storage for communication disks and service group heartbeat disks.

Preexisting Network Partitions

A *preexisting network partition* refers to failures in communication channels that occur while the systems are down. Regardless of whether the cause is scheduled maintenance or system failure, VCS cannot respond to failures when systems are down. This leaves VCS vulnerable to network partitioning when the systems are booted.

VCS Seeding

To protect your cluster from a preexisting network partition, VCS employs the concept of a *seed*. By default, when a system comes up, it is not *seeded*. Systems can be seeded automatically or manually.

Note that only systems that have been seeded can run VCS.

Systems are seeded automatically in one of two ways:

- When an unseeded system communicates with a seeded system.
- When all systems in the cluster are unseeded and able to communicate with each other.

VCS requires that you declare the number of systems that will participate in the cluster. When the last system is booted, the cluster will seed and start VCS on all systems. Systems can then be brought down and restarted in any combination. Seeding is automatic as long as at least one instance of VCS is running somewhere in the cluster.

Manual seeding is required only to run VCS from a cold start (all systems down) when not all systems are available.

VCS Installation Procedures

The remainder of this chapter provides step-by-step instructions for setting up and configuring your hardware and installing VCS. It also describes the hardware and software requirements for the cluster, and explains various options available to you.

Installing VCS involves various tasks, and each task must be performed in the order presented below. Detailed instructions begin on page 17.

- ✓ Set up the hardware (page 17).
- ✓ Install VCS (page 22).
- ✓ Configure the VCS communication services (page 23).
- ✓ Allocate storage for communication disks and service group heartbeat disks (page 27).
- ✓ Start VCS (page 30).
- ✓ Verify LLT, GAB, and Cluster operation (page 31).
- ✓ Initialize file systems and disk groups on shared storage (page 36).
- ✓ Prepare for NFS services (page 37).

After completing these procedures, you are ready to configure VCS. Refer to the *VERITAS Cluster Server User's Guide* to complete VCS configuration.

Setting Up the Hardware

Requirements

Item	Description
VCS systems	SPARC systems running Solaris 2.5.1 or later.
CD-ROM drive	One CD-ROM drive on each system, or a drive accessible to each.
Disks	Typical configurations require shared disks to support applications that migrate between systems in the cluster.
Ethernet controllers	VCS requires at least one Ethernet controller per system, in addition to the built-in public Ethernet controller.
SCSI adapters	VCS requires at least one built-in SCSI adapter per system for the operating system disks, and at least one additional SCSI adapter per system for shared data disks.
Disk space	Each VCS system must have at least 35 megabytes of free space in the `/opt` file system.
RAM	Each VCS system requires at least 128 megabytes.

Configuring the Network and Storage

1. Install the required Ethernet and SCSI controllers.
2. Connect the VCS private Ethernet controllers on each system. Use cross-over Ethernet cables (supported only on two systems), or independent hubs for each VCS communication network.
3. To probe your shared SCSI buses and select IDs for the systems, cable the shared devices to one system and terminate the SCSI bus.

   a. From the `EEPROM` prompt (`ok`) type the following commands:

   ```
   ok show-devs
   ok probe-scsi-all
   ```

   b. Select a unique SCSI ID for each system on the shared SCSI bus.
   The priority of SCSI IDs is 7 to 0, followed by 15 to 8. Use high-priority IDs for the systems, and low-priority IDs for devices such as disks and tape drives. For example, use 7, 6, 5, and so forth, for the systems, and use the remaining IDs for devices.

4. Modify the configuration file of the shared SCSI driver on each system.

   a. Identify the three-letter name and device parent for the shared SCSI controller. Type the following command on the system connected to the shared devices:

   ```
   # ls -l shared_disk_partition
   ```

   The variable *shared_disk_partition* refers to the device path for any device on the shared SCSI bus.

   For example, type:

   ```
   # ls -l /dev/dsk/c1t0d0s3
   ```

   Output resembles:

   ```
   lrwxrwxrwx 1 root root 53 Dec 03 11:10 \
      /dev/dsk/c1t0d0s3 -> ../../ \
      devices/sbus@1f,0/QLGC,isp@0,10000/sd@0,0:d,raw
   ```

   Common SCSI driver names include `isp` (in bold text in the preceding sample output) and `fas` (not shown).

Note that the parent name `/sbus@1f,0` (also in bold text in the preceding sample output) includes the slash after the word `devices`, and extends to, but does not include, the slash preceding the driver name.

b. Identify the register property values for the shared SCSI controller. Type the following command on the system connected to the shared devices:

```
# prtconf -v
```

Output resembles:

```
QLGC,isp, instance #0
...
Register Specifications:
    Bus Type=0x0, Address=0x10000, Size=1c2
...
```

c. Modify `/kernel/drv/`*driver_name*`.conf` on each system to set the SCSI ID for the system to use on the shared bus. (Create this file if it does not exist.)

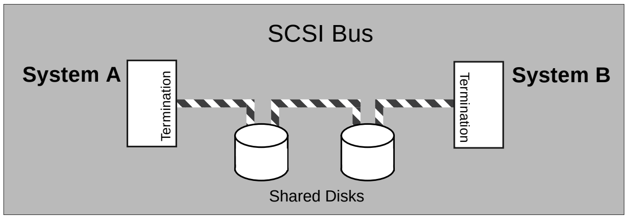
```
name="driver_name" parent="parent_name"
reg=register_property_values
scsi-initiator-id=scsi_id;
```

For example, the file `/kernel/drv/isp.conf` for the system with SCSI ID 5 would resemble:

```
name="isp" parent="/sbus@1f,0"
reg=0x0,0x10000,1c2
scsi-initiator-id=5;
```

5. Shut down all systems in the cluster.

6. Cable the shared devices. Two-system clusters can be cabled with the devices between them, as illustrated in the figure below:



To cable shared devices on more than two systems:

Disable SCSI termination on systems that are not positioned at the ends of the SCSI chain. Most single-ended SCSI controllers auto-detect to disable termination, or are configured through software. Differential SCSI controllers typically require that you remove resistors from the controller card. For more information, refer to the documentation on controllers configured in your systems.

7. Boot each system:

```
ok boot -r
```

Watch for console messages from the driver changing the SCSI ID.
For example:

```
isp0: initiator SCSI ID now 5
```

8. Use the `prtvtoc`(1M) command to test the connection from each system to the shared devices.

For example, type:

```
# prtvtoc /dev/dsk/c1t0d0s0
```

Output from this command confirms the connection between the system on which you typed the command and the disk.

Network Partitions and the Sun Boot Monitor

Sun SPARC systems provide a console-abort sequence that enables you to halt and continue the processor:

- `L1-A` or `STOP-A` on the keyboard
- `BREAK` on the serial console input device.

Each command is then followed by a response of `go` at the `ok` prompt. Continuing operations after the processor has stopped may corrupt data and is therefore unsupported by VCS. Specifically, when a system is halted with the abort sequence it stops producing heartbeats. The other systems in the cluster then consider the system failed and take over its services. If the system is later enabled with `go`, it continues writing to shared storage as before, even though its applications have been restarted on other systems.

Solaris 2.6

In Solaris 2.6, Sun introduced support for disabling the abort sequence. We recommend disabling the console-abort sequence on systems running Solaris 2.6 or greater. To do this:

1. Add the following line to the `/etc/default/kbd` file (create the file if it does not exist):

   ```
   KEYBOARD_ABORT=disable
   ```

2. Reboot.
3. If necessary, refer to the `kbd`(1) manual page for details.

Solaris 2.5.1

If you do not want to disable the abort sequence, do not type `go` at the `EEPROM` prompt (`ok`). If a system has been stopped with the abort sequence in a VCS cluster, type `boot` at the `ok` prompt.

Installing VCS

On each system, perform the following steps to install the VCS software.

1. Insert the CD into a drive connected to your system.
   - If you are running Solaris volume-management software, the software automatically mounts the CD as `/cdrom/cdrom0`.
   - If you are not running Solaris volume-management software, you must mount the CD manually. For example:

     ```
     # mount -F hsfs -o ro /dev/dsk/c0t6d0s2 \
          /cdrom/cdrom0
     ```

     Note that `/dev/dsk/c0t6d0s2` is the default name for the CD drive.

2. Add the VCS packages.

   a. On each system, type the following command to install the VCS software:

   ```
   # pkgadd -d /cdrom/cdrom0
   ```

   You will receive a message listing the available packages.

   b. When prompted, select `all`.

3. As the packages are being added, answer `Yes` when prompted.

4. After all the packages are added, type `q`. The following message is displayed:

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

CAUTION! Do not reboot at this time. Before rebooting, you must configure the VCS communication services, LLT and GAB. Proceed to the next section, “Configuring the VCS Communication Services.”

Configuring the VCS Communication Services

VCS communication services include two kernel components: LLT and GAB.

- LLT (Low Latency Transport) provides fast, kernel-to-kernel communications, and monitors network connections.
- GAB (Group Membership and Atomic Broadcast) provides the global message order required to maintain a synchronized state, and monitors disk communications.

LLT and GAB are used only by VCS. They replace the functions of TCP/IP for VCS private network communications. LLT and GAB provide the performance and reliability required by VCS for these and other functions.

LLT and GAB must be configured as described in the following sections.

Configuring Low Latency Transport (LLT)

To configure LLT, set up an `/etc/llttab` configuration file on each system in the cluster.

Each `/etc/llttab` file must specify the system's ID number, the network interfaces to use, and other directives. Refer to the sample `llttab` file in `/opt/VRTSllt`.

The following example shows a simple `llttab` with minimum directives.

```
set-node 1
link qfe0 /dev/qfe:0 - ether - -
link qfe1 /dev/qfe:1 - ether - -
start
```

These and other directives used in the `/etc/llttab` configuration file are described on page 24. For more information on LLT directives, refer to the `llttab`(4) manual page.

LLT Directives

`set-node`	Assigns the system ID. This number must be unique for each system in the cluster, and must be in the range 0-31. *Note that LLT fails to operate if any systems share the same ID*.
`link`	Attaches LLT to a network interface. At least one link is required, and up to eight are supported. The first argument to `link` is a user-defined tag shown in the `lltstat`(1M) output to identify the link. It may also be used in `llttab` to set optional static MAC addresses. The second argument to `link` is the device name of the network interface. Its format is *device_name:device_instance_number*. The remaining four arguments to link are defaults; these arguments should be modified only in advanced configurations. There should be one `link` directive for each network interface. LLT uses an unregistered Ethernet SAP of 0xCAFE. If the SAP is unacceptable, refer to the `llttab`(4) manual page for information on how to customize SAP. Note that IP addresses do not need to be assigned to the network device; LLT does not use IP addresses.
`start`	The `start` directive must always appear last.
`set-cluster`	Assigns a unique cluster number. Use this directive when more than one cluster is configured on the same physical network connection. Note that LLT uses a cluster number of zero.
`link-lowpri`	Use this directive in place of `link` for public network interfaces. This directive prevents VCS communication on the public network until the network is the last link, and reduces the rate of heartbeat broadcasts. Note that LLT distributes network traffic evenly across all available network connections and, in addition to enabling VCS communication, broadcasts heartbeats to monitor each network connection.

Note: The order of directives must be the same as in the sample file, `/opt/VRTSllt/llttab`.

Additional Considerations for LLT

- Each network interface configured for LLT must be attached to a separate and distinct physical network.
- By default, Sun systems assign the same MAC address to all interfaces. Thus, connecting two or more interfaces to a network switch can cause problems. For example, if IP is configured on one public interface and LLT on another, and both interfaces are connected to a switch, the duplicate MAC address on the two switch ports can cause the switch to incorrectly redirect IP traffic to the LLT interface and vice-versa. To avoid this, configure the system to assign unique MAC addresses by setting the `eeprom`(1M) parameter `local-mac-address` to `true` (when using network switches instead of network hubs).

Configuring Group Membership and Atomic Broadcast (GAB)

To configure GAB, set up an `/etc/gabtab` configuration file on each system in the cluster.

Each `/etc/gabtab` file must specify the number of systems in the cluster. Refer to the sample `gabtab` file in `/opt/VRTSgab`.

The following example shows a simple `gabtab`:

```
gabconfig -c -n 2
```

This configuration has no communication disks and expects two systems to join before VCS is seeded automatically.

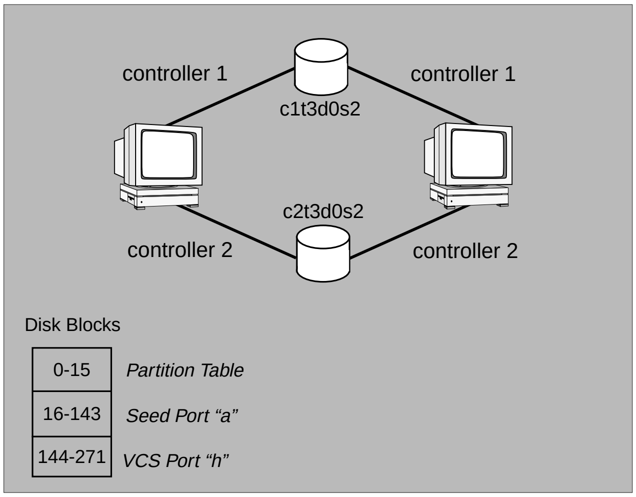
To bypass protection from preexisting network partitions, replace the `-n` option with the `-x` option in the `gabconfig` command in `/etc/gabtab` on all systems. The resulting `gabtab` file is:

```
gabconfig -c -x
```

Configuring Communication Disks

To configure communication disks, add `gabdisk` commands to the `/etc/gabtab` configuration file on each system in the cluster. Each `gabdisk` command must specify the device name, the start location, and the GAB port used for the communication disk.

In the following illustration, two systems are connected by two shared I/O chains that resolve to controllers 1 and 2 in Solaris device names. For example, partition 2 of disk target number 3 could be allocated for VCS communication on both shared I/O chains. Because partition 2 begins on the first cylinder of the disk, the regions must start on or after block 16.



Allocation of Communication Disk Regions

This configuration is specified in the `/etc/gabtab` file, which resembles:

```
gabdisk -a /dev/dsk/c1t3d0s2 -s 16 -p a
gabdisk -a /dev/dsk/c1t3d0s2 -s 144 -p h
gabdisk -a /dev/dsk/c2t3d0s2 -s 16 -p a
gabdisk -a /dev/dsk/c2t3d0s2 -s 144 -p h
gabconfig -c -n 2
```

The `-s` option to the command `gabdisk` specifies the start location of each 128-block region. The `-p` option specifies the port, the value `a` specifies the seed port, and the value `h` specifies the VCS port. If the partition is not the first partition on the disk, the start locations are 0 and 128, and only 256 blocks of the partition are required.

Allocating Storage for Communication Disks and Service Group Heartbeat Disks

Each communication disk or service group heartbeat disk refers to a 64K (128-block) region of a physical disk partition. These regions are specified by a block device name and block offset. Multiple regions can be configured in different partitions on the same physical disk, and in the same partition at different, non-overlapping offsets.

A communication disk requires two physical disk regions, one for seeding and one for VCS. If there are multiple I/O chains connected to all systems, we recommend configuring a communication disk on each chain. Each disk requires two 128-block regions. If the partition begins in the first cylinder of the disk, avoid blocks 0-15 to leave room for the partition table.

Communication disks are specified with identical `gabdisk` commands in `/etc/gabtab` on each system in the cluster. Service group heartbeat disks are configured as a VCS resource. (Refer to the “ServiceGroupHB Agent” on page 56.)

Configuring Disk Regions on Volume Manager Disks

Communication disk regions and service group heartbeat disk regions can coexist on a disk controlled by VERITAS Volume Manager (VxVM). However, these disk regions cannot be configured on VxVM volumes, and must be configured instead on the block ranges of the underlying physical device. The space for these partitions must be allocated before a disk is initialized by the Volume Manager.

Follow the steps below to prepare a disk for VCS communication and Volume Manager storage:

1. Install VxVM as instructed in the *VERITAS Volume Manager Installation and Configuration Guide*.
2. Identify the disk by its VxVM tag name, for example, `c1t1d0`.

3. If the disk contains data, migrate the data to another storage media.
   a. Unmount all file systems on the disk.
   b. Remove any volumes, plexes, or subdisks from the disk.
   c. Remove the disk from any active disk group or deport its disk group.
4. On any system, place the VCS command directory in your path.
   For example:
   ```
   # export PATH=$PATH:/opt/VRTSvcs/bin
   ```
5. Allocate a VCS partition on the disk. Type:
   ```
   # hahbsetup disk_tag
   ```
   Enter `y` when prompted. The `hahbsetup` command sets up disk communication for VxVM and VCS. The variable *disk_tag* refers to the name you identified in step 2. For example:
   ```
   # hahbsetup c1t1d0
   ```
   Output resembles:
   ```
   The hadiskhb command is used to set up a disk for combined use
   by VERITAS Volume Manager and VERITAS Cluster Server for disk
   communication.

   WARNING: This utility will destroy all data on c1t1d0

   Have all disk groups and file systems on disk c1t1d0 been either
   unmounted or deported? y

   There are currently slices in use on disk /dev/dsk/c1t1d0s2

   Destroy existing data and reinitialize disk? y

   1520 blocks are available for VxCS disk communication and
   service group heartbeat regions on device /dev/dsk/c1t1d0s7

   This disk can now be configured into a Volume Manager disk
   group. Using vxdiskadm, allow it to be configured into the disk
   group as a replacement disk. Do not select reinitialization of
   the disk.

   After running vxdiskadm, consult the output of prtvtoc to
   confirm the existence of slice 7. Reinitializing the disk
   under VxVM will delete slice 7. If this happens, deport the disk
   group and rerun hahbsetup.
   ```

6. The disk should now be initialized, even though it has not been added to a disk group. To add the disk to a disk group, run the `vxdg addisk` command (refer to the `vxdg` manual page for more information). For example, after running `hahbsetup` to allocate a VCS partition on `clt1d0`, add `clt1d0` to the sharedg disk group as `disk01` by typing the following command:

   ```
   # vxdg -g sharedg adddisk disk01=clt1d0
   ```

7. Display the partition table. Type:

   ```
   # prtvtoc /dev/dsk/disk_tags0
   ```

   For example:

   ```
   # prtvtoc /dev/dsk/c1t1d0s0
   ```

   Output resembles:

```
Partition     Tag  Flags        First      Sector       Last   Mount  Directory
                               Sector       Count     Sector
        2       5     01            0     8887440    8887439
        3      15     01            0        1520       1519
        4      14     01         3040     8884400    8887439
        7      13     01         1520        1520       3039
```

8. Confirm that slice 7 exists and that its tag is 13.

9. Configure partition `/dev/dsk/c1t1d0s7` into VCS.

Starting VCS

To start VCS, reboot each system in the cluster.

During the reboot process, console messages about LLT and GAB are displayed.

If LLT and GAB are configured correctly on each system, the output resembles:

```
.
.
.
LLT: link 0 node 1 active
LLT: link 1 node 1 active
.
.
.
VCS: starting on: thor17
VCS: waiting for configuration status
VCS: local configuration missing
VCS: registering for cluster membership
VCS: waiting for cluster membership
GAB: Port h gen 19fb0003 membership 01
VCS: received new cluster membership
VCS: all systems have stale configurations
.
.
.
```

Note that the links are `active` for LLT and that `Port h` is registered for GAB; therefore, LLT and GAB are configured correctly. Now, you must verify that the cluster is operating. Refer to “Verifying LLT, GAB, and Cluster Operation” on page 31.

If your network is not performing correctly, the output resembles:

```
.
.
.
VCS: starting on: thor17
VCS: waiting for configuration status
VCS: local configuration missing
VCS: registering for cluster membership
GAB: Port h registration waiting for seed port membership
VCS: registration failed. Exiting
.
.
.
```

Note that `Port h` registration is incomplete. The network devices or cabling are configured incorrectly. Now, you must examine LLT, GAB, and the cluster for potential problems. Refer to “Verifying LLT, GAB, and Cluster Operation,” below.

Verifying LLT, GAB, and Cluster Operation

Before attempting to verify LLT, GAB, or the cluster, you must:

✓ Log in to any system in the cluster as `root`.

✓ Place the VCS command directory in your path as instructed in step 4 on 28.

Verifying LLT

Use the `lltstat` command to verify that links are active for LLT. This command returns information about the links for LLT for the system on which it is typed. Refer to the `lltstat`(4) manual page for more information.

In the following example, `lltstat -n` is typed on each system in a private network.

System 1

```
# lltstat -n
```

Output resembles:

```
LLT node information:
    Node        State       Links
  * 1           OPEN        2
    2           OPEN        2
sys1#
```

System 2

```
# lltstat -n
```

Output resembles:

```
LLT node information:
    Node        State       Links
    1           OPEN        2
  * 2           OPEN        2
sys2#
```

Note that each system has two links and that each system is in the `OPEN` state. The asterisk (*) denotes the system on which the command is typed.

If the output of `lltstat -n` does not show all the systems in the cluster or two links for each system, use the verbose option of `lltstat`. For example, type `lltstat -nvv | more` on a system to view additional information about LLT.

In the following example, `lltstat -nvv | more` is typed on a system in a private network.

System 1

```
# lltstat -nvv | more
```

Output resembles:

```
Node      State        Link     Status   Address
0         CONNWAIT
                       qfe0     DOWN
                       qfe1     DOWN
* 1       OPEN
                       qfe0     UP       08:00:20:93:0E:34
                       qfe1     UP       08:00:20:93:0E:34
2         OPEN
                       qfe0     UP       08:00:20:8F:D1:F2
                       qfe1     UP       08:00:20:8F:D1:F2
3         CONNWAIT
                       qfe0     DOWN
                       qfe1     DOWN
```

Note that each system should be `OPEN`, each link should be `UP`, and each address should be correct. If the output of `lltstat` indicates otherwise, LLT is not operating. In this case, LLT is not configured correctly.

To obtain information about the ports open for LLT, type `lltstat -p` on a system. In the following example, `lltstat -p` is typed on a system in a private network.

System 1

```
# lltstat -p
```

Output resembles:

```
LLT port information:
    Port    Usage       Cookie
    0       gab         0x0
         opens:      0 1 3 4 5 6 7 8 9 10 11 12 13...
         connects:   1 2
sys1#
```

Note that two systems (1 and 2) are connected.

Verifying GAB

To verify that GAB is operating, type the following command as `root` on each system:

```
# /sbin/gabconfig -a
```

If GAB is operating, the following GAB port membership information is returned:

```
GAB Port Memberships
===================================
Port a gen a36e0003 membership 01
Port h gen fd570002 membership 01
```

`Port a` indicates that Gab is communicating; `gen a36e0003` is a random generation number; `membership 01` indicates that systems `0` and `1` are connected.

`Port h` indicates that VCS is started; `gen fd570002` is a random generation number; `membership 01` indicates that systems `0` and `1` are both running VCS.

If GAB is not operating, no GAB port membership information is returned:

```
GAB Port Memberships
===================================
```

If only one network is connected, the following GAB port membership information is returned:

```
GAB Port Memberships
===================================
Port a gen a36e0003 membership 01
Port a gen a36e0003 jeopardy    1
Port h gen fd570002 membership 01
Port h gen fd570002 jeopardy    1
```

For more information on GAB, refer to the *VERITAS Cluster Server User’s Guide*.

Verifying the Cluster

To verify that the cluster is operating, type the following command as `root` on one system:

```
# /opt/VRTSvcs/bin/hasys -display
```

Output resembles:

```
#System   Attribute         Value
thor17    AgentsStopped     0
thor17    ConfigBlockCount  0
thor17    ConfigCheckSum    0
thor17    ConfigDiskState   INVALID
thor17    ConfigFile        /etc/VRTSvcs/conf/config
thor17    ConfigInfoCnt     0
thor17    ConfigModDate     Wed Dec 31 16:00:00 1969
thor17    DiskHbDown
thor17    Frozen            0
thor17    GUIIPAddr
thor17    LinkHbDown
thor17    Load              0
thor17    LoadRaw           runque 0 memory 0 disk 0...
thor17    MajorVersion      1
thor17    MinorVersion      8
thor17    NodeId            0
thor17    OnGrpCnt          0
thor17    SourceFile        ./main.cf
thor17    SysName           thor17
thor17    SysState          STALE_ADMIN_WAIT
thor17    TFrozen           0
thor17    UserInt           0
thor17    UserStr
```

Note the system state (`SysState`) attribute value.

If LLT and GAB are operating and the `SysState` attribute value is `STALE_ADMIN_WAIT`, the system is operating and waiting to transition to a `RUNNING` state.

Note that VCS cannot run because it does not have a `main.cf` file in the `/etc/VRTSvcs/conf/config` directory.

The `main.cf` file is not automatically installed with the `VRTSvcs` package. This file is created in the VCS configuration procedure. Refer to the *VERITAS Cluster Server User's Guide*.

At this point in the installation process, VCS is successfully installed.

Now, stop VCS on all systems. Type:

```
# /opt/VRTSvcs/bin/hastop -all
```

For more information on the `hasys -display` and `hastop -all` commands, or on VCS system states, refer to the *VERITAS Cluster Server User's Guide*.

Initializing File Systems and Disk Groups on Shared Storage

In addition to the shared disk partitions used for VCS communications, your configuration may include disks on the shared bus that contain VERITAS Volume Manager disk groups or file systems. You must initialize these disk groups and file systems from one system only.

For VxVM configurations, install VxVM as instructed in the *VERITAS Volume Manager Installation Guide*. Disks on the shared bus must be configured into disk groups other than `rootdg`. Create disk groups on one system only. They will be deported and imported onto the other system by VCS as necessary. Similarly, use `mkfs` to make shared file systems from one system only. They will be mounted on other systems by VCS as necessary.

Note: Do not add exported file systems to `/etc/vfstab` or `/etc/dfs/dfstab`. VCS will mount and export these file systems automatically.

Preparing NFS Services

Identifying Block Devices

Your configuration may include disks on the shared bus that support NFS. File systems exported by NFS can be configured on disk partitions or on VERITAS Volume Manager volumes.

An example disk partition name is `/dev/dsk/c1t1d0s3`. An example volume name is `/dev/vx/dsk/shareydg/vol3`. Each name represents the block device on which the file system will be mounted.

Checking Major and Minor Numbers

Block devices providing NFS service must have the same major and minor numbers on each system. Major and minor numbers are used by Solaris to identify the logical partition or disk slice. NFS also uses them to identify the exported file system. Major and minor numbers must be checked to ensure that the NFS identity for the file system is the same when exported from each system.

To check major and minor numbers:

1. Use the following command on all systems that are exporting an NFS file system. This command displays the major and minor numbers for the block device. For Volume Manager volumes, you must first import the associated shared disk group on each system.

   ```
   # ls -lL block_device
   ```

   The variable *block_device* refers to a partition on which a file system is mounted for export via NFS.

   This procedure must be followed for each NFS file system, one at a time.

   For example, type:

   ```
   # ls -lL /dev/dsk/c1t1d0s3
   ```

   Output on System A resembles:

   ```
   crw-r-----  1 root  sys  32,134 Dec 3 11:50 /dev/dsk/c1t1d0s3
   ```

Output on System B resembles:

```
  crw-r-----  1 root  sys  32,134 Dec 3 11:55 /dev/dsk/c1t1d0s3
```

Note that the major and minor numbers shown above match; both major numbers are 32, and both minor numbers are 134.

2. If either the major or minor numbers do not match, refer to “Chapter 3, “Troubleshooting VCS.”
3. Repeat steps 1–2 for each block device used for NFS.

VCS Resource Types and Agents

2

Agents are VCS processes that perform functions according to commands received from VCS.

VCS agents:

- ✓ Bring resources online.
- ✓ Take resources offline.
- ✓ Monitor resources and report any state changes to VCS.

A system has one agent per resource type that monitors all resources of that type; for example, a single IP agent manages all IP resources.

When the agent is started, it pulls the necessary configuration information from VCS. It then periodically monitors the resources, and updates VCS with the resource status.

Enterprise Agents

Enterprise agents are not included with the VCS software, but are sold separately. Contact your VERITAS sales representative for details on these agents or additional agents under development:

- Informix
- NetBackup
- Oracle
- Sybase

Bundled Agents

The agents described in the remainder of this chapter are included with the VCS software, and are referred to as *bundled* agents:

- Disk
- DiskGroup
- FileOnOff
- IP
- IPMultiNIC
- Mount
- MultiNICA
- NFS
- NIC
- Phantom
- Process
- Proxy
- ServiceGroupHB
- Share
- Volume

Note: The CLARiiON agent for CLARiiON SCSI disk array split-bus configurations is no longer supported. The CLARiiON SCSI disk array, series 1000, in dual-bus configurations continues to be supported. For more information, see the *VCS 1.1 Release Notes*.

Disk Agent

Description	Manages a raw disk.	
Entry Points	• *Online*—Not applicable. • *Offline*—Not applicable. • *Monitor*—Determines if disk is accessible by performing read I/O on raw disk.	
Required Attribute	**Type and Dimension**	**Definition**
`Partition`	string-scalar	Indicates which partition to monitor. Partition is specified with the full path beginning with a slash (/). Otherwise the name given is assumed to reside in `/dev/rdsk`.

Type Definition

```
type Disk (
      str Partition
      NameRule = resource.Partition
      static str Operations = None
      static str ArgList[] = { Partition }
)
```

Sample Configuration

```
Disk c1t0d0s0 (
      Partition = c1t0d0s0
)
```

DiskGroup Agent

Description	Brings online, takes offline, and monitors a VERITAS Volume Manager disk group.	
Entry Points	• *Online*—Using the command `vxdg`, this script imports the disk group. • *Offline*—Using the command `vxdg`, this script deports the disk group. • *Monitor*—Using the command `vxdg`, this agent determines if the disk group is online or offline. If disk group has been imported with `noautoimport=off`, and if the group is not frozen, the group to which the disk group belongs is taken offline.	
Required Attribute	**Type and Dimension**	**Definition**
`DiskGroup`	string-scalar	Disk group name.
Optional Attributes	**Type and Dimension**	**Definition**
`StartVolumes`	string-scalar	If value is 1, the DiskGroup `online` script starts all volumes belonging to that disk group after importing. Default is 1.
`StopVolumes`	string-scalar	If value is 1, the DiskGroup `offline` script stops all volumes belonging to that disk group before deporting. Default is 1.

Type Definition

```
type DiskGroup (
      static int OnlineRetryLimit = 1
      str DiskGroup
      NameRule = resource.DiskGroup
      static str ArgList[] = { DiskGroup, StartVolumes,
                                  StopVolumes, MonitorOnly }
      str StartVolumes = 1
      str StopVolumes = 1
      static int NumThreads = 1
)
```

Sample Configuration

```
DiskGroup sharedg (
      DiskGroup = sharedg
)
```

FileOnOff Agent

Description	Creates, removes, and monitors files.	
Entry Points	• *Online*—Creates an empty file with the specified name, if one does not already exist. • *Offline*—Removes the specified file. • *Monitor*—Checks if the specified file exists. If it does, the agent reports as online. If it does not, the agent reports as offline.	
Required Attribute	**Type and Dimension**	**Definition**
`PathName`	string-scalar	Specifies the complete pathname, starting with the slash (/) preceding the file name.

Type Definition

```
type FileOnOff (
      str PathName
      NameRule = resource.PathName
      static str ArgList[] = { PathName }
)
```

Sample Configuration

```
FileOnOff tmp_file01 (
      PathName = "/tmp/file01"
)
```

IP Agent

Description	Manages the process of configuring an IP address on an interface.	
Entry Points	• *Online*—Checks if the IP address is in use by another system. Uses `ifconfig` to set the IP address on a unique alias on the interface. • *Offline*—Brings down the IP address associated with the specified interface. Uses `ifconfig` to set the interface alias to `0.0.0.0` and the state to “down.” • *Monitor*—Monitors the interface to test if the IP address associated with the interface is alive.	
Required Attributes	**Type and Dimension**	**Definition**
`Address`	string-scalar	IP address associated with the interface.
`Device`	string-scalar	Name of the NIC resource associated with the IP address. Should only contain the resource name without an alias; for example, `le0`.
Optional Attributes	**Type and Dimension**	**Definition**
`ArpDelay`	integer-scalar	Number of seconds to sleep between configuring an interface and sending out a broadcast to inform routers about this IP address. Default is 1 second.
`IfconfigTwice`	integer-scalar	Causes an IP address to be configured twice, using an `ifconfig` `up-down-up` sequence. Increases probability of gratuitous `arps` (caused by `ifconfig up`) reaching clients. Default is 0.
`NetMask`	string-scalar	Netmask associated with the interface.
`Options`	string-scalar	Options for the `ifconfig` command.

Type Definition

```
type IP (
      str Device
      str Address
      str NetMask
      str Options
      int ArpDelay = 1
      int IfconfigTwice = 0
      NameRule = resource.Address
      static str ArgList[] = { Device, Address, NetMask,
                                Options, ArpDelay, IfconfigTwice
                             }
)
```

Sample Configuration

```
IP    IP_192_203_47_61 (
        Device = le0
        Address = "192.203.47.61"
)
```

Mount Agent

Description	Brings online, takes offline, and monitors a file system mount point.	
Entry Points	• *Online*—Mounts a block device on the directory. If mount fails, the agent tries to run `fsck` on the raw device to remount the block device. • *Offline*—Unmounts the file system. • *Monitor*—Determines if the file system is mounted. Checks mount status using the commands `stat` and `statvfs`.	
Required Attributes	**Type and Dimension**	**Definition**
`BlockDevice`	string-scalar	Block device for mount point.
`MountPoint`	string-scalar	Directory for mount point.
`FSType`	string-scalar	File system type, for example, vxfs, ufs, etc.
Optional Attributes	**Type and Dimension**	**Definition**
`FsckOpt`	string-scalar	Options for `fsck` command.
`MountOpt`	string-scalar	Options for `mount` command.

Type Definition

```
type Mount (
      str MountPoint
      str BlockDevice
      str FSType
      str MountOpt
      str FsckOpt
      NameRule = resource.MountPoint
      static str ArgList[] = { MountPoint, BlockDevice,
                                FSType, MountOpt, FsckOpt }
)
```

Sample Configuration

```
Mount export1 (
      MountPoint= "/export1"
      BlockDevice = "/dev/dsk/c1t1d0s3"
      FSType = vxfs
      MountOpt = ro
)
```

NFS Agent

Description	Starts and monitors the `nfsd` and `mountd` processes required by all exported NFS file systems.
Entry Points	• *Online*—Checks if `nfsd` and `mountd` processes are running. If they're not, it starts them and exits. • *Offline*—Not applicable. • *Monitor*—Monitors versions 2 and 3 of the `nfsd` process, and versions 1, 2, and 3 of the `mountd` process. Monitors `tcp` and `udp` versions of the processes by sending RPC (Remote Procedure Call) calls `clnt_create` and `clnt_call` to the RPC server. If calls succeed, the resource is reported as online.

Optional Attribute	**Type and Dimension**	**Definition**
`Nservers`	integer-scalar	Specifies the number of concurrent NFS requests the server can handle. Default is 16.

Type Definition

```
type NFS (
        int Nservers = 16
        NameRule = "NFS_" + group.Name + "_" + resource.Nservers
        static str ArgList[] = { Nservers }
        static str Operations = OnOnly
        static int RestartLimit = 1
)
```

Sample Configuration

```
NFS NFS_groupx_24 (
        Nservers = 24
)
```

NIC Agent

Description	Monitors the configured NIC. If a network link fails, or if there is a problem with the device card, the resource is marked as offline. The NIC listed in the `Device` attribute must have an administration IP address, which is the default IP address assigned to the physical interface of a host on a network. The agent will not configure network routes or an administration IP address.	
Entry Points	• *Online*—Not applicable. • *Offline*—Not applicable. • *Monitor*—Tests the network card and network link. Uses the DLPI (Data Link Provider Interface) layer to send and receive messages across the driver. Pings the broadcast address of the interface to generate traffic on the network. Counts the number of packets passing through the device before and after the address is pinged. If the count decreases or remains the same, the resource is marked as offline.	
Required Attribute	**Type and Dimension**	**Definition**
`Device`	string-scalar	NIC name.
Optional Attributes	**Type and Dimension**	**Definition**
`PingOptimize`	integer-scalar	Number of monitor cycles to detect if configured interface is inactive. A value of 1 optimizes broadcast pings and requires two monitor cycles. A value of 0 performs a broadcast ping during each monitor cycle and detects the inactive interface within the cycle. Default is 1.
`NetworkHosts`	string-vector	List of hosts on the network that will be pinged to determine if the network connection is alive. The IP address of the host should be entered instead of the HostName to prevent the monitor from timing out (DNS problems can cause the ping to hang); for example, `166.96.15.22`. If this optional attribute is not specified, the monitor will test the NIC by pinging the broadcast address on the NIC. If more than one network host is listed, the monitor will return online if at least one of the hosts is alive.
`NetworkType`	string-scalar	Type of network, such as Ethernet (ether), FDDI (fddi), Token Ring (token), etc.

Type Definition

```
type NIC (
      str Device
      str NetworkType
      str NetworkHosts[]
      NameRule = group.Name + "_" + resource.Device
      int PingOptimize = 1
      static str ArgList[] = { Device, NetworkType,
                                 NetworkHosts, PingOptimize }
      static str Operations = None
)
```

Sample Configurations

Sample 1: Without Network Hosts, Using the Default Ping Mechanism

```
NIC   groupx_le0 (
           Device = le0
           PingOptimize = 1
           )
```

Sample 2: With Network Hosts

```
NIC   groupx_le0 (
           Device = le0
           NetworkHosts = { "166.93.2.1", "166.99.1.2" }
           )
```

Phantom Agent

Description	Enables groups with no OnOff resources to display the correct status. (Include Phantom resource types in parallel groups only.) Service groups that do not include OnOff resources as members are not brought online, even if their member resources are brought online, because the status of the None and OnOnly resources are not considered when deciding if a group is online.
Entry Point	• *Online*—Not applicable. • *Offline*—Not applicable. • *Monitor*—Determines status based on the status of its group.

Type Definition

```
type Phantom (
     static str ArgList[] = {}
     NameRule = Phantom_ + group.Name
)
```

Sample Configuration

```
Phantom (
)
```

Process Agent

Description	Starts, stops, and monitors a process specified by the user.	
Entry Points	• *Online*—Starts the process with optional arguments. • *Offline*—VCS sends a `SIGTERM`. If the process does not exit within one second, VCS sends a `SIGKILL`. • *Monitor*—Checks to see if the process is alive by scanning the process table for the name of the executable pathname and argument list. Because of the Solaris `procfs`-interface, the match is limited to the initial 80 characters.	
Required Attribute	**Type and Dimension**	**Definition**
`PathName`	string-scalar	Defines complete pathname for accessing an executable program, including the program name.
Optional Attribute	**Type and Dimension**	**Definition**
`Arguments`	string-scalar	Passes arguments to the process. **Note:** Multiple tokens must be separated by one space only. String cannot accommodate more than one space between tokens, or leading or trailing whitespace characters.

Type Definition

```
type Process (
      str PathName
      str Arguments
      NameRule = resource.PathName
      static str ArgList[] = { PathName, Arguments }
)
```

Sample Configuration

```
Process   usr_lib_sendmail (
      PathName = /usr/lib/sendmail
      Arguments = "bd q1h"
)
```

Proxy Agent

Description	Mirrors the state of another resource on the local or a remote system.	
Entry Point	• *Online*—Not applicable. • *Offline*—Not applicable. • *Monitor*—Determines status based on the target resource status.	
Required Attribute	**Type and Dimension**	**Definition**
`TargetResName`	string-scalar	Name of the target resource whose status is mirrored by Proxy resource.
Optional Attribute	**Type and Dimension**	**Definition**
`TargetSysName`	string-scalar	Status of `TargetResName` on system mirrored by Proxy resource. If attribute is not specified, Proxy resource assumes the system is local.

Type Definition

```
type (
      static str ArgList[] = { TargetResName, TargetSysname,
                                 "TargetResName:Probed",
                                 TargetResName:State }
      NameRule = Proxy_ + resource.TargetResName
      static str Operations = None
      str TargetResName
      str TargetSysName
)
```

Sample Configurations

Sample 1

```
// Proxy resource to mirror the state of the resource
// tmp_VRTSvcs_file1 on the local system.
Proxy(
      TargetResName = "tmp_VRTSvcs_file1"
)
```

Sample 2

```
// Proxy resource to mirror the state of the resource
// tmp_VRTSvcs_file1 on sys1.
Proxy (
      TargetResName = "tmp_VRTSvcs_file2"
      TargetSysName = "sys1"
)
```

ServiceGroupHB Agent

Description	Starts, stops, and monitors disk heartbeats associated with service groups. (See the *VERITAS Cluster Server User’s Guide* for details.)
Entry Points	• *Online*—Starts a kernel process that periodically writes heartbeats to disk. It first monitors the disk to ensure that no other system is writing to it. Requires that all disks be available before sending heartbeats. • *Offline*—Stops the heartbeat process for this disk partition. • *Monitor*—Verifies that the heartbeat process is active by reading the heartbeats from disk. Requires only one disk to remain online.

Required Attribute	**Type and Dimension**	**Definition**
`Disks`	string-vector	Specifies, in paired values, the disk partition and the block location to use for the heartbeat. If the same partition is used for more than one heartbeat, block numbers must be 64k (128) apart. Note that a block device partition is used for the disk heartbeating; for example, `/dev/dsk/clt2d0s3`.

Type Definition

```
type ServiceGroupHB (
      str Disks[]
      NameRule = SGHB_ + resource.Disks
      static str ArgList[] = { Disks }
)
```

Sample Configuration

```
ServiceGroupHB    SGHB_c1t2d0s3 (
      Disks = { c1t2d0s3, 0, c2t2d0d3, 0 }
)
```

Share Agent

Description	Shares, unshares, and monitors a single local resource for exporting an NFS file system to be mounted by remote systems.	
Entry Points	• *Online*—Shares an NFS file system. • *Offline*—Unshares an NFS file system. • *Monitor*—Reads `/etc/dfs/sharetab` file and looks for an entry for the file system specified by `PathName`. If the entry exists, it returns as online.	
Required Attribute	**Type and Dimension**	**Definition**
`PathName`	string-scalar	Pathname of the file system to be shared.
Optional Attributes	**Type and Dimension**	**Definition**
`OfflineNFSRestart`	integer-scalar	Restarts NFS when the `offline` entry point is executed. Default is 1. If there are multiple shares in a single service group, setting this attribute for one share only is sufficient.
`OnlineNFSRestart`	integer-scalar	Restarts NFS when the `online` entry point is executed. Default is 0. If there are multiple shares in a single service group, setting this attribute for one share only is sufficient.
`Options`	string-scalar	Options for the `share` command.

Type Definition

```
type Share (
      str PathName
      str Options
      int OnlineNFSRestart = 0
      int OfflineNFSRestart = 1
      NameRule = nfs + resource.PathName
      static str ArgList[] = { PathName, Options,
                                OnlineNFSRestart,
                                OfflineNFSRestart }
)
```

Sample Configuration

```
Share nfsshare1x (
      PathName = "/share1x"
)
```

Volume Agent

Description	Brings online, takes offline, and monitors a VERITAS Volume Manager volume.	
Entry Points	• *Online*—Using the command `vxvol`, this agent starts the volume. • *Offline*—Using the command `vxvol`, this agent stops the volume. • *Monitor*—Determines if the volume is online or offline by reading a block from the raw device interface to the volume.	
Required Attributes	**Type and Dimension**	**Definition**
`DiskGroup`	string-scalar	Disk group name.
`Volume`	string-scalar	Volume name.

Type Definition

```
type Volume (
      str Volume
      str DiskGroup
      NameRule = resource.DiskGroup + "_" + resource.Volume
      static str ArgList[] = { Volume, DiskGroup }
)
```

Sample Configuration

```
Volume sharedg_vol3 (
      Volume = vol3
      DiskGroup = sharedg
)
```

Additional Agents

VCS provides two additional agents that enable you to configure failover between NICs on a single system: IPMultiNIC and MultiNICA. These agents are described in the following pages. For configurations using a single NIC, use the IP and NIC agents described on page 45 and page 50, respectively.

IPMultiNIC Agent

Description	Represents a virtual IP address configured as an alias on one interface of a MultiNICA resource. If multiple service groups have IPMultiNICs associated with the same MultiNICA resource, only one group will have the MultiNICA resource. The other groups will have Proxy resources pointing to it.	
Entry Points	• *Online*—Configures a virtual IP address on one interface of the MultiNICA resource. • *Offline*—Removes a virtual IP address from one interface of the MultiNICA resource. • *Monitor*—Checks if the virtual IP address is configured on one interface of the MultiNICA resource.	
Required Attributes	**Type and Dimension**	**Definition**
`Address`	string-scalar	Virtual IP address.
`MultiNICResName`	string-scalar	Name of associated MultiNICA resource.
Optional Attributes	**Type and Dimension**	**Definition**
`ArpDelay`	integer-scalar	Number of seconds to sleep between configuring an interface and sending out a broadcast to inform routers about this IP address. Default is 1 second.
`IfconfigTwice`	integer-scalar	Causes an IP address to be configured twice, using an `ifconfig up-down-up` sequence. Increases probability of gratuitous `arps` (caused by `ifconfig up`) reaching clients. Default is 0.
`NetMask`	string-scalar	NetMask for the virtual IP address. Default is “+”.
`Options`	string-scalar	The `ifconfig` options for the virtual IP address.

Type Definition

```
    type IPMultiNIC (
            static str ArgList[] = { "MultiNICResName:Device",
                  Address, NetMask, "MultiNICResName:ArpDelay",
                  Options, "MultiNICResName:Probed",
                  MultiNICResName, ArpDelay, IfconfigTwice }
            NameRule = IPMultiNIC_ + resource.Address
            str Address
            str NetMask
            str Options
            str MultiNICResName
            int ArpDelay = 1
            int IfconfigTwice = 0
            static int MonitorTimeOut = 120
        )
```

Sample Configuration

```
  group grp1 (
            SystemList = { sysa, sysb }
            AutoStartList = { sysa }
            )
    MultiNICA mnic (
            Device@sysa = { le0 = "166.98.16.103",qfe3 = "166.98.16.103" }
            Device@sysb = { le0 = "166.98.16.104",qfe3 = "166.98.16.104" }
            NetMask = 255.255.255.0
            ArpDelay = 5
            Options = "trailers"
            RouteOptions@sysa = "default 166.98.16.103 0"
            RouteOptions@sysb = "default 166.98.16.104 0"
    )
```

```
IPMultiNIC ip1 (
      Address = "166.98.14.78"
      NetMask = "255.255.255.0"
      MultiNICResName = mnic
      Options = "trailers"
)
ip1 requires mnic
group grp2 (
      SystemList = { sysa, sysb }
      AutoStartList = { sysa }
      )
IPMultiNIC ip2 (
      Address = "166.98.14.79"
      NetMask = "255.255.255.0"
      MultiNICResName = mnic
      Options = "trailers"
)
Proxy proxy (
      TargetResName = mnic
)
ip2 requires proxy
```

MultiNICA Agent

Description	Represents a set of network interfaces and provides failover capabilities between them. Each interface in a MultiNICA resource has a base IP address, which can be the same or different. The MultiNICA agent configures one interface at a time. If it does not detect activity on the configured interface, it configures a new interface and migrates IP aliases to it. If an interface is associated with a MultiNICA resource, it should not be associated with any other MultiNIC or NIC resource. If the same set of interfaces must be a part of multiple service groups, configure a MultiNICA resource in one of the service groups, and Proxy resources that point to the MultiNIC resource in the other service groups.	
Entry Point	• *Online*—Not applicable. • *Offline*—Not applicable. • *Monitor*—Checks for activity on a configured interface by sampling input packets received on that interface. If it does not detect activity, it forces activity by sending out a broadcast ping. If it detects a failure, it migrates to the next interface.	
Required Attribute	**Type and Dimension**	**Definition**
`Device`	string-association	List of interfaces and their base IP addresses.
Optional Attributes	**Type and Dimension**	**Definition**
`ArpDelay`	integer-scalar	Number of seconds to sleep between configuring an interface and sending out a broadcast to inform routers about base IP address. Default is 1 second.
`Handshake-Interval`	integer-scalar	Helps compute the number of times the monitor will ping the network host after migrating to a new NIC. Default value is 90.
`IfconfigTwice`	integer-scalar	Causes an IP address to be configured twice, using an `ifconfig up-down-up` sequence. Increases probability of gratuitous `arps` (caused by `ifconfig up`) reaching clients. Default is 0.
`NetMask`	string-scalar	Netmask for the base IP address. Default is “+”.

`NetworkHosts`	string-vector	List of hosts on the network that will be “pinged” to determine if the network connection is alive. The IP address of the host should be entered instead of the HostName to prevent the monitor from timing out (DNS will cause the ping to hang). If this optional attribute is not specified, the monitor will test the NIC by pinging the broadcast address on the NIC. If more than one network host is listed, the monitor will return online if at least one of the hosts is alive (for example, `NetworkHosts = {“166.93.2.1”,“166.97.1.2”}`).
`Options`	string-scalar	The `ifconfig` options for the base IP address; for example, “trailers.”
`PingOptimize`	integer-scalar	Number of monitor cycles to detect if configured interface is inactive. A value of 1 optimizes broadcast pings and requires two monitor cycles. A value of 0 performs a broadcast ping each monitor cycle and detects the inactive interface within the cycle. Default is 1.
`RouteOptions`	string-scalar	String to add a route when configuring an interface. This string contains the information “destination gateway metric.” No routes are added if this string is set to `NULL`.

Note: If all the NICs configured in the `Device` attribute are down, the MultiNICA agent will fault the resource after a 2-3 minute interval. This delay occurs because the MultiNICA agent tests the failed NIC several times before marking the resource offline. Messages recorded in the engine log (`/var/VRTSvcs/log/engine.log_A`) during failover provide a detailed description of the events that take place during failover.

Type Definition

```
    type MultiNICA (
            static str ArgList[] = { Device, NetMask, ArpDelay,
                  Options, RouteOptions, PingOptimize,
                  MonitorOnly, IfconfigTwice, HandshakeInterval,
                  NetworkHosts }
            static str Operations = None
            static int MonitorTimeout = 300
            NameRule = MultiNICA_ + group.Name
            str Device{}
            str NetMask
            int ArpDelay = 1
            str Options
            str RouteOptions
            int PingOptimize = 1
            int IfconfigTwice = 0
            int HandshakeInterval = 90
            str NetworkHosts[]
        )
```

Sample Configuration

```
  group grp1 (
            SystemList = { sysa, sysb }
            AutoStartList = { sysa }
            )
    MultiNICA mnic (
            Device@sysa = { le0 = "166.98.16.103",qfe3 = "166.98.16.103" }
            Device@sysb = { le0 = "166.98.16.104",qfe3 = "166.98.16.104" }
            NetMask = "255.255.255.0"
            ArpDelay = 5
            Options = "trailers"
            RouteOptions@sysa = "default 166.98.16.103 0"
            RouteOptions@sysb = "default 166.98.16.104 0"
    )
    IPMultiNIC ip1 (
            Address = "166.98.14.78"
            NetMask = "255.255.255.0"
            MultiNICResName = mnic
            Options = "trailers"
    )
    ip1 requires mnic
  group grp2 (
            SystemList = { sysa, sysb }
            AutoStartList = { sysa }
            )
    IPMultiNIC ip2 (
            Address = "166.98.14.79"
            NetMask = "255.255.255.0"
            MultiNICResName = mnic
            Options = "trailers"
    )
    Proxy proxy (
            TargetResName = mnic
    )
    ip2 requires proxy
```

Troubleshooting VCS 3

Reconciling Major and Minor Numbers

Block devices providing NFS service must have the same major and minor numbers on each system. Major and minor numbers are used by Solaris to identify the logical partition or disk slice. NFS also uses them to identify the exported file system. These numbers must be checked to ensure that the NFS identity for the file system is the same when exported from each system.

1. Use the following command to display the major and minor numbers for each of the block devices supporting NFS services from each system.
   For Volume Manager volumes, you must first import the associated shared disk group on each system, then type the following command:

   # ls -lL *block_device*

   For example, type:

   ```
   # ls -lL /dev/dsk/c1t1d0s3
   ```

   Output from each system resembles:

   On System A:

   ```
   crw-r-----  1 root  sys  32,134 Dec 3 11:50 /dev/dsk/c1t1d0s3
   ```

   On System B:

   ```
   crw-r-----  1 root  sys  36, 62 Dec 3 11:55 /dev/dsk/c1t1d0s3
   ```

2. If the major numbers match, proceed to step 4. (In the example above, the major numbers are 32 and 36.)

3. If they do not match, complete steps a-f, below.

   a. Place the VCS command directory in your path. For example:

      ```
      # export PATH=$PATH:/opt/VRTSvcs/bin
      ```

   b. If the block device is a volume, identify on each system the two major numbers used by the VERITAS Volume Manager:

      ```
      # grep vx /etc/name_to_major
      ```

      Output on System A would resemble:

      ```
      vxio 32
      vxspec 33
      ```

      On System B:

      ```
      vxio 36
      vxspec 37
      ```

   c. Type the following command on System B to change the major number (36/37) to match that of System A (32/33):

      For disk partitions:

      ```
      # haremajor -sd major_number
      ```

      For volumes:

      ```
      # haremajor -vx major_number1 major_number2
      ```

      The variable *major_number* represents the numbers from System A.

      For example, for disk partitions:

      ```
      # haremajor -sd 32
      ```

      For volumes:

      ```
      # haremajor -vx 32 33
      ```

      If this command fails, you will receive a report similar to the following:

      ```
      Error: Preexisiting major number 32
      These are available numbers on this system: 128...
      Check /etc/name_to_major on all systems for
      available numbers.
      ```

d. If you receive this report, type the following command on System A to change the major number (32/33) to match that of System B (36/37):

   For disk partitions:

   ```
   # haremajor -sd 36
   ```

   For volumes:

   ```
   # haremajor -vx 36 37
   ```

   If the command fails again, you will receive a report similar to the following:

   ```
   Error: Preexisiting major number 36
   These are available numbers on this node: 126...
   Check /etc/name_to_major on all systems for
   available numbers.
   ```

e. If you receive the second report, choose the larger of the two available numbers (in this example, 128), and use this number in the `haremajor` command to reconcile the major numbers. Type the following command on both systems:

   For disk partitions:

   ```
   # haremajor -sd 128
   ```

   For volumes:

   ```
   # haremajor -vx 128 129
   ```

f. Reboot each system on which `haremajor` was successful.

4. If the minor numbers match, proceed to reconcile the major and minor numbers of your next partition.

5. If the block device on which the minor number does not match is a volume, consult the manual page `vxdg(1M)` for instructions on reconciling the VERITAS Volume Manager minor numbers, with specific reference to the `reminor` option.

6. For disk partitions, complete steps a–e, below. (In this example, the minor numbers are 134 and 62.)

   a. Type the following command on both systems using the name of your block device:

      ```
      # ls -1 /dev/dsk/c1t1d0s3
      ```

      Output from this command resembles the following on System A:

      ```
      lrwxrwxrwx  1 root  root  83 Dec 3 11:50 \
         /dev/dsk/c1t1d0s3 -> ../../ \
         devices/sbus@1f,0/QLGC,isp@0,1000/sd@2,0:d,raw
      ```

      The device name (in bold, above) includes the slash following the word `devices`, and continues to, but does not include, the colon.

   b. Type the following command on both systems to determine the instance numbers used by the SCSI driver:

      ```
      # grep sd /etc/path_to_inst | sort -n -k 2,2
      ```

Output from this command resembles the following on System A:

```
"/sbus@1f,0/QLGC,isp@0,10000/sd@0,0" 0 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@1,0" 1 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@2,0" 2 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@3,0" 3 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@4,0" 4 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@5,0" 5 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@6,0" 6 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@8,0" 7 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@9,0" 8 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@a,0" 9 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@b,0" 10 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@c,0" 11 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@d,0" 12 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@e,0" 13 "sd"
"/sbus@1f,0/QLGC,isp@0,10000/sd@f,0" 14 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@0,0" 15 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@1,0" 16 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@2,0" 17 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@3,0" 18 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@4,0" 19 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@5,0" 20 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@6,0" 21 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@8,0" 22 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@9,0" 23 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@a,0" 24 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@b,0" 25 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@c,0" 26 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@d,0" 27 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@e,0" 28 "sd"
"/sbus@1f,0/SUNW,fas@e,8800000/sd@f,0" 29 "sd"
```

c. Locate the device names in the output of step b, and identify the instance numbers that appear as the second field in each line. In this example, the device name on System A is `/sbus@1f,0/QLGC,isp@0,10000/sd@1,0`. The associated instance number is `1`.

d. Compare instance numbers.

- If the instance number from one system is not used on the other (that is, it does not appear in the output of step b), edit `/etc/path_to_inst` to make the second system's instance number equal to that of the first system.
- If the instance numbers are being used on both systems, edit `/etc/path_to_inst` on both systems. Change the instance number associated with the device name to an unused number greater than the highest number used by other devices. The output of step b shows the instance numbers used by all devices.

e. Type the following command to reboot each system on which `/etc/path_to_inst` was modified:

```
# reboot -- -rv
```

Index