SGI® InfiniteStorage High Availability Using Linux®-HA Heartbeat

007–5451–002

New Features in this Guide

This book has been updated to support the SGI InfiniteStorage Software Platform (ISSP) 1.4 release. It contains the following changes:

- Support for CXFS. See:
  - "High Availability and SGI Products" on page 1
  - "CXFS Requirements" on page 10
  - Figure 4-1 on page 19
  - "Standard CXFS" on page 23
  - "Configuring CXFS for HA " on page 32
  - "Testing the `cxfs` Resource" on page 34
- Support for the Pacemaker cluster resource manager

Record of Revision

Version	Description
001	July 2008 Original publication, supporting the SGI InfiniteStorage Software Platform (ISSP) 1.3 release
002	September 2008 Revised to support ISSP 1.4 release

Contents

Figures

Tables

Examples

About This Guide

This publication provides information about creating resources for the high-availability (HA) SGI resource agents provided for the Linux-HA Heartbeat product.

Prerequisites

To use this guide, you must understand the information about configuring and using Linux-HA Heartbeat provided by the following:

- The *Heartbeat* manual provided by Novell, Inc., in the SUSE Linux Enterprise Server 10 (SLES 10) `sles-heartbeat_en` RPM and on the following website:

  http://www.novell.com/documentation/sles10

- Linux-HA project website:

  http://linux-ha.org/

Note: The external websites referred to in this guide were correct at the time of publication, but are subject to change.

Related Publications

In addition to the Heartbeat documentation discussed above in "Prerequisites," the following SGI publications contain additional information:

- *CXFS 5 Administration Guide for SGI InfiniteStorage*
- *DMF Administrator's Guide for SGI InfiniteStorage*
- *DMF Filesystem Audit Guide for SGI InfiniteStorage*
- *OpenVault Operator's and Administrator's Guide*
- *SGI L1 and L2 Controller Software User's Guide*
- *SGI Altix 450 System User's Guide*

- *SGI Altix 4700 System User's Guide*
- *SGI Altix XE240 System User's Guide*
- *SGI Altix XE250 System User's Guide*
- *SGI Altix XE310 System User's Guide*
- *SGI Altix XE320 System User's Guide*
- *TMF Administrator's Guide*
- *TMF Release and Installation Guide*
- *TMF User's Guide*
- *XVM Volume Manager Administrator's Guide*

Obtaining Publications

You can obtain SGI documentation as follows:

- See the SGI Technical Publications Library at http://docs.sgi.com. Various formats are available. This library contains the most recent and most comprehensive set of online books, release notes, man pages, and other information.
- View release notes on your system by accessing the `README.txt` file for the product. This is usually located in the `/usr/share/doc/`*productname* directory, although file locations may vary.
- You can view man pages by typing `man` *title* at a command line.

Conventions

In this guide, *Heartbeat* refers to the Linux-HA Heartbeat product.

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.

variable	Italic typeface denotes variable entries and words or concepts being defined.
user input	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. (Output is shown in nonbold, fixed-space font.)
[ ]	Brackets enclose optional portions of a command or directive line.
...	Ellipses indicate that a preceding element can be repeated.
manpage(*x*)	Man page section identifiers appear in parentheses after man page names.
GUI	This font denotes the names of graphical user interface (GUI) elements such as windows, screens, dialog boxes, menus, toolbars, icons, buttons, boxes, fields, and lists.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this publication, contact SGI. Be sure to include the title and document number of the publication with your comments. (Online, the document number is located in the front matter of the publication. In printed publications, the document number is located at the bottom of each page.)

You can contact SGI in any of the following ways:

- Send e-mail to the following address:

  techpubs@sgi.com

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

- Send mail to the following address:

  SGI
  Technical Publications
  1140 East Arques Avenue
  Sunnyvale, CA 94085–4602

SGI values your comments and will respond to them promptly.

Introduction

This chapter discusses the following:

High Availability and SGI Products

The Linux-HA Heartbeat product provides the infrastructure to fail over *highly available (HA) resources* that survive a single point of failure. A *resource* is a service, associated with an IP address, that is managed by Heartbeat. A *resource group* is a set of resources that must be managed and failed over from one node to another as a set. Heartbeat starts, monitors, and stops resources. A *resource agent* is the set of software that allows a service to be highly available without modifying the application itself.

Heartbeat uses the IP address of a resource to redirect clients to the node currently running the resource. Each resource is actively owned by one node. If that node fails, an alternate node restarts the HA applications of the failed node. To application clients, the services on the alternate node are indistinguishable.

SGI provides the following resource agents:

Resource Agent	Description
`cxfs`	CXFS clustered filesystem
`dmf`	Data Migration Facility (DMF)
`lxvm`	Local XVM volume manager
`openvault`	OpenVault mounting service for DMF
`tmf`	Tape Management Facility (TMF) mounting service for DMF

`l2network`	STONITH (*"shoot the other node in the head"*) system reset for SGI Altix systems with L1/L2 controllers
`sgi-ipmi`	STONITH system reset for SGI Altix XE x86_64 systems with a baseboard management controller (BMC) using intelligent platform management interface (IPMI) network reset

This guide leads you through a procedure to include the SGI resource agents in a Heartbeat cluster, using as an example a resource group named `dmfGroup` that contains all of the required component resources (such as `lxvm`, `Filesystem`, `tmf`, `dmf`, and so on). Figure 1-1 shows the concept of failing over the `dmfGroup` resource group from `node-0` to `node-1`.

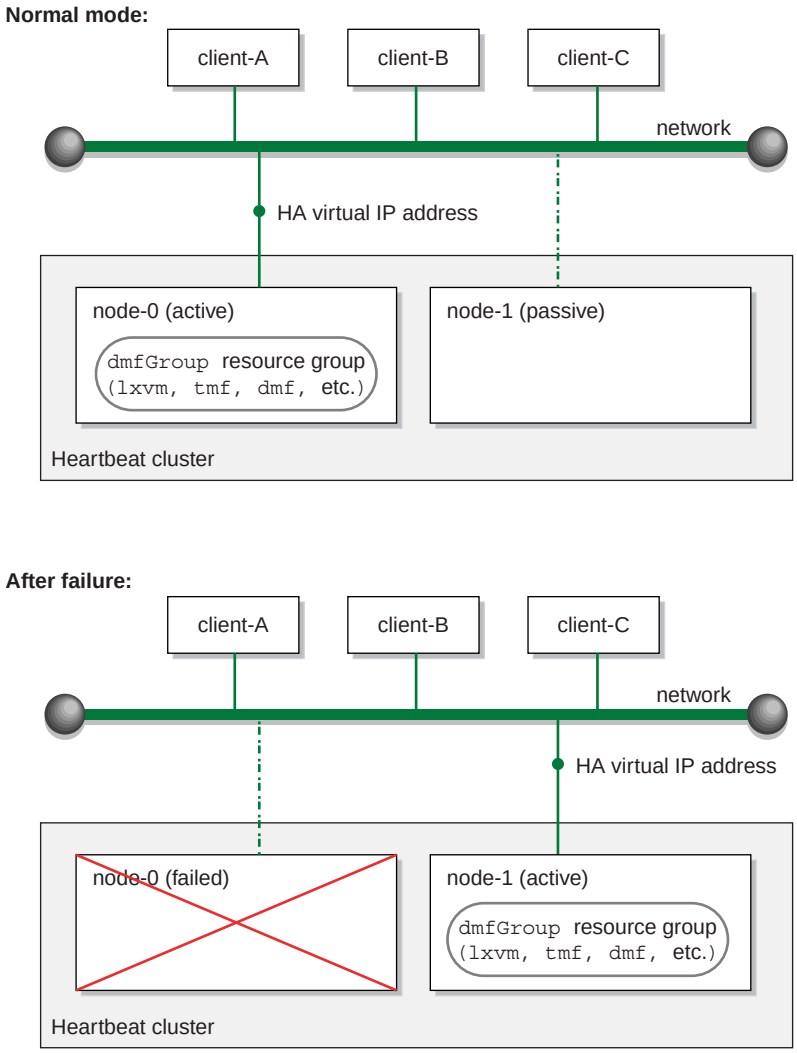

Figure 1-1 Resource Group Failover

This guide does not discuss high availability (HA) in general, nor provide details about configuring a Heartbeat cluster. For a brief overview of the changes you must make, see Chapter 6, "Establishing a Heartbeat Cluster" on page 27. For information

about software installation, see the *SGI InfiniteStorage Software Platform* (ISSP) release note.

Software Packages

SGI provides repackaging of the 2.1.3 community version of the Linux-HA Heartbeat product with additional fixes (which will also be pushed back to the Linux community) as part of the ISSP software distribution. The following RPMs make up the ISSP HA distribution:

RPM	Contents
`sgi-heartbeat`	Heartbeat tools
`sgi-heartbeat-common`	Heartbeat infrastructure and STONITH reset infrastructure, and the resource agent for the STONITH IPMI reset service
`sgi-heartbeat-resources`	Open Cluster Framework (OCF) resources
`sgi-heartbeat-plugins`	The `cxfs`, `dmf`, `lxvm`, `openvault`, and `tmf`, resource agents
`sgi-heartbeat-l1l2`	Resource agent for the STONITH L2 reset service
`sgi-pacemaker`	Pacemaker cluster resource manger

Heartbeat Configuration Tools

The procedures in this guide use XML files and the `cibadmin` command to modify the Heartbeat cluster information base (CIB).

You could also use the `hb_gui` graphical user interface (GUI). You may need to add the user ID under which you choose to run the `hb_gui` to the `haclient` group in `/etc/group`. When you log in to `hb_gui`, you must supply this user ID and its system login password.

Caution: Although you will use YaST to initially install ISSP software, you cannot use YaST to configure Heartbeat for an SGI cluster because the SGI cluster uses the `sgi-heartbeat` package and YaST requires the `heartbeat` package. Attempting to use YaST to configure your Heartbeat cluster may leave your system in an unsupported configuration by replacing SGI's Heartbeat software.

When establishing dependencies among resources, you can either use specific Heartbeat colocation and ordering restraints or create a resource group and add the resources to the resource group in the correct dependency order. The procedures in this guide use the simpler resource-group model.

For more information about these commands, see "Sources for Detailed Heartbeat Documentation" on page 5.

Sources for Detailed Heartbeat Documentation

For details about Heartbeat, see the following:

- The *Heartbeat* guide provided by Novell, Inc. in the SUSE Linux Enterprise Server 10 (SLES 10) `sles-heartbeat_en` RPM and at the following website:[1]

  http://www.novell.com/documentation/sles10

- Linux-HA project website:
  - http://linux-ha.org/
  - http://linux-ha.org/v2/AdminTools/cibadmin
  - http://linux-ha.org/GuiGuide
  - http://linux-ha.org/HeartbeatResourceAgent/IPaddr2
  - http://linux-ha.org/ha.cf/UcastDirective
  - http://linux-ha.org/LSB
  - http://linux-ha.org/LSBResourceAgent
  - http://linux-ha.org/OCF

1 The external websites referred to in this guide were correct at the time of publication, but are subject to change.

- http://linux-ha.org/OCFResourceAgent
- http://linux-ha.org/ScoreCalculation

Note: You will not use the installation information provided by the Novell or Linux-HA project.

Best Practices for High Availability

The following are best practices when using SGI high-availability (HA) resource agents:

- Fix networking issues first.
- Make your overall system configuration as simple as possible — complexity makes HA harder to achieve.
- Use redundancy in your system components to avoid single points of failure.
- Perform regular and frequent system backups.
- Configure and test the standard services (like DMF) in normal mode before making them highly available — doing so will make problems easier to diagnose.
- Configure and test the base Heartbeat cluster before adding the SGI resources.
- Review the overall procedure in Chapter 4, "Outline of the Configuration Procedure for SGI High-Availability Products" and then follow the detailed steps in Chapter 5, "Configuring and Testing the Standard Services" through Chapter 8, "Configuring and Testing STONITH Reset Services".
- Copy-and-paste the XML examples from this book as a starting point, replacing bold values with values appropriate for your site. (You should not change other values unless you have good reason to do so and are aware of the implications.)
- Use resource groups to simplify configuration of colocation and ordering restraints.

Note: Resource group names and `Filesystem` resource names must not have spaces in them if they are to contain an OpenVault resource. In general, it is not a good idea to use spaces in resource names as this may cause Heartbeat or other supporting software to behave in a confusing manner.

- Define `start`, `stop`, and `monitor` operations for resources as shown in this guide. Specify time values in seconds, which requires adding `s` to numerical values (otherwise, the default is milliseconds).
- Always use STONITH (“*shoot the other node in the head*”) reset services to protect data integrity in case of failure.

- Always add Heartbeat fencing as a prerequisite to `start` operations.
- Always use `fence` as the `on_fail` action for all `stop` operations. (*Fencing* in Linux-HA Heartbeat terminology is not the same as *fencing* in CXFS terminology.)
- Only set the debugging level in `/etc/ha.d/ha.cf` to a value greater than `1` if you are actively debugging the system; values `2` and larger can have a negative effect on the logfile size and on cluster reliability. Appropriate messages will be sent to the `/var/log/messages` file using the default debugging level.
- For a cluster of two nodes, use `ucast` for communication (which sends UDP datagrams directly to each node). For more information, see:

  http://linux-ha.org/ha.cf/UcastDirective
- You may want to examine the use of *resource stickiness* to control how often and to what nodes failover will occur. The examples in this book result in the resource group failing over to the alternate node on the first failure of any of the resources in the resource group. In this default setting, there is no automatic failback to the first node.

  For more information about score calculation, see the documentation referred to in "Sources for Detailed Heartbeat Documentation" on page 5 and the following website:

  http://linux-ha.org/ScoreCalculation
- Before making changes to an existing Heartbeat configuration, first make a backup copy of the cluster information base (CIB) so that you can return to it if necessary. See "Making a Backup Copy of the CIB" on page 78.
- Periodically watch the output of the following commands for problems:

  ```
  # crm_mon -r1
  # crm_verify -LV
  ```

  Refer to the Heartbeat logs if errors or warnings are noticed, and periodically to ensure that you are aware of operations automatically initiated by Heartbeat. See "Reviewing Log Files" on page 78.

Preliminary Requirements for High Availability

This chapter discusses the following requirements for high availability (HA):

Licensing Requirements

All nodes in an SGI Linux-HA Heartbeat cluster must have the appropriate software licenses installed. The following software requires licenses if used:

- CXFS
- DMF (plus associated licenses for APD and either the OpenVault or TMF mounting service)
- Enhanced NFS
- XVM snapshot

For information about obtaining licenses, see the individual product administration guides.

Software Version Requirements

For any of the SGI resource agents, you must use the corresponding version of SGI software as defined in the *SGI InfiniteStorage Software Platform* release note.

Hardware Requirements

Due to STONITH reset requirements, all nodes in an HA cluster should be of the same system type. SGI supports an HA cluster that consists of one of the following system types:

- SGI Altix XE240
- SGI Altix XE250
- SGI Altix XE310
- SGI Altix XE320
- SGI Altix 450
- SGI Altix 4700

Note: As an exception, you can mix SGI Altix 450 and SGI Altix 4700 systems within one cluster.

This release supports DMF running on two nodes in active/passive mode.

System Reset Requirements

You must use STONITH (“*shoot the other node in the head*”) reset services to protect data integrity in case of failure. See Chapter 8, "Configuring and Testing STONITH Reset Services" on page 71.

CXFS Requirements

The CXFS resource agent allows you to associate other products with the failover of the CXFS metadata server in a Linux-HA Heartbeat cluster. This requires the CXFS

resource agent and the corresponding version of CXFS software, as defined in the *SGI InfiniteStorage Software Platform* release note. This section discusses the following:

Number of Nodes in the CXFS Heartbeat Cluster

The Heartbeat cluster must include every CXFS server-capable administration node for every filesystem that is managed by the CXFS resource agent. CXFS client-only nodes may also be members of the Heartbeat cluster if they manage resources that are dependent upon CXFS (such as Apache or an FTP server). However, you must add location constraints to the Heartbeat cluster information base (CIB) so that the CXFS resource agent will be started only on the CXFS server-capable administration nodes.

The CXFS server-capable administration nodes must use a CXFS fail policy of `reset`. You should otherwise configure the CXFS cluster, nodes, and filesystems according to the instructions in the following:

CXFS 5 Administration Guide for SGI InfiniteStorage
CXFS 5 Client-Only Guide for SGI InfiniteStorage

Choosing CXFS Nodes for Failover

You must give careful consideration when choosing nodes used for the Heartbeat cluster. Certain resources (such as DMF), require that the CXFS metadata server and the Heartbeat resource be provided by the same node; see "DMF Requirements" on page 14. Other resources (such as NFS and Samba) do not have this requirement, but it may be desirable to enforce it in order to ensure that these resources provide the best performance possible. (Some NFS and Samba workloads can cause significant performance problems when the NFS or Samba resource is provided from a node that is not also the CXFS metadata server.)

Unless a resource must run on the CXFS metadata server, you should configure Heartbeat so that it does not relocate the CXFS metadata server when it fails-over a resource.

CXFS Relocation Support

CXFS relocation is provided automatically by the CXFS resource agent. In a CXFS cluster running Heartbeat, relocation should only be started by using the tools provided with Heartbeat and not by use of any other method. See "CXFS Requirements" on page 10.

Applications that Depend Upon CXFS Filesystems

You must use co-location and start-ordering constraints or ordered resource groups so that every application using a CXFS filesystem and running on a metadata server that is managed by Heartbeat must also be managed by Heartbeat. You must set co-location and start-ordering constraints such that:

- The application will not run on a server-capable administration node that is not the active CXFS metadata server for the filesystem that it uses
- The CXFS metadata server will start before the application starts and stop after the application stops

Nodes and System Reset

Nodes must system reset in order to prevent conflicts with CXFS I/O fencing methods. For more information, see Chapter 8, "Configuring and Testing STONITH Reset Services" on page 71.

CXFS Start/Stop Issues

You should start CXFS cluster services and CXFS services before starting Heartbeat. The CXFS resource agent will wait for all of the CXFS filesystems to be mounted by CXFS before attempting any relocation. You must adjust the `start` operation timeout for the CXFS resource agent accordingly.

During failover, resources that co-locate with the CXFS metadata server must be stopped before the CXFS resource. If a resource fails to shutdown completely, any

files left open on the metadata server will prevent relocation. Therefore, the Heartbeat fail policy for any resource that could prevent relocation by holding files open must be `fence` and you must configure a STONITH facility according to the requirements for your site. See Chapter 8, "Configuring and Testing STONITH Reset Services" on page 71

In this case, the offending CXFS metadata server will be reset, causing recovery to the alternate node.

Local XVM Requirements

All local XVM volumes that are managed by Heartbeat must have unique `volname` values.

All local XVM physical volumes (*physvols*) that are managed by Heartbeat must have unique `Disk Name` values in their XVM label when compared to all other XVM volumes on the SAN. For example, you cannot have two physvols on the same SAN with the `Disk Name` of `spool` even if one is foreign.

If you do not have unique values, potential problems are:

- Heartbeat stealing the wrong physvol from a system outside of the cluster while I/O is ongoing and losing data from that system while corrupting the filesystem from the node within the cluster by whom it is stolen
- General confusion in Heartbeat, resulting in node reset

OpenVault Requirements

If OpenVault is to be used as the DMF mounting service, you must do the following:

- Provide a directory for OpenVault's use within an HA filesystem in the DMF resource group. This is known as the *serverdir directory* (as specified in "Configuring OpenVault for HA" on page 47). The directory will hold OpenVault's database and logs. The directory can be either of the following:
  - Within the root of a Heartbeat-managed filesystem dedicated for OpenVault use
  - Within another Heartbeat-managed filesystem, such as the filesystem specified by the DMF configuration file `HOME_DIR` parameter

In non-HA configurations, the OpenVault server's files reside in `/var/opt/openvault/server`. During the conversion to HA, OpenVault will move its databases and logs into the specified directory within a Heartbeat-managed filesystem and change `/var/opt/openvault/server` to be a symbolic link to that directory.

- Create a virtual hostname for use by OpenVault and either add it to your local DNS server or add it to the `/etc/hosts` file on each host in the cluster that will be used as a DMF server or as an OpenVault client node. The address associated with that virtual hostname must be a virtual address managed by a community `IPaddr2` resource within the same resource group as the `openvault` resource. You may also use the `IPaddr2` virtual address for other purposes, such as NFS. For more information, see "Configuring a Virtual IP Address for HA" on page 44.

TMF Requirements

There are no special requirements to run TMF in an HA environment.

DMF Requirements

Using DMF with Heartbeat requires the following:

- The Heartbeat cluster must contain all of the nodes that will be DMF servers. (The cluster can contain additional nodes that will not be DMF servers so long as there are constraints on the DMF resource group such that it will run only on the appropriate subset of cluster nodes. For more information about these constraints, see the documentation referred to in "Sources for Detailed Heartbeat Documentation" on page 5.)
- Each DMF server must run the required product and HA software.
- All DMF server nodes in the Heartbeat cluster must have connectivity to all of the XFS filesystems that DMF either depends upon or manages. All of the local XVM volumes that make up those filesystems must be managed by an `lxvm` resource within the same resource group as the `dmf` resource. Each of the filesystems must be managed by a community `Filesystem` resource in that resource group.

The DMF filesystems to be managed are:

– The DMF-managed user filesystems

– DMF administrative filesystems specified by the following parameters in the DMF configuration file:

```
HOME_DIR
JOURNAL_DIR
SPOOL_DIR
TMP_DIR
MOVE_FS
```
`CACHE_DIR` for any Library Servers
`STORE_DIRECTORY` for any DCMs and disk MSPs using local disk storage

DMF requires independent paths to tape drives so that they are not fenced by CXFS. The ports for the tape drive paths on the switch should be masked from fencing in a CXFS configuration.

The SAN must be zoned so that XVM does not failover CXFS filesystem I/O to the paths visible through the tape HBA ports when Fibre Channel port fencing occurs. Therefore, either independent switches or independent switch zones should be used for CXFS/XVM volume paths and DMF tape drive paths.

For more information about DMF filesystems, see the *DMF Administrator's Guide for SGI InfiniteStorage*.

- All DMF server nodes in the Heartbeat cluster must have connectivity to the same set of tape libraries and drives. If a node only had access to a subset of the drives, and the DMF server is failed over to that node, DMF would then not be able to access data on tapes left mounted in inaccessible drives.
- The ordering of resources within a resource group containing a `dmf` resource must be as noted in step 8 of Chapter 4, "Outline of the Configuration Procedure for SGI High-Availability Products" on page 17.

Outline of the Configuration Procedure for SGI High-Availability Products

This chapter summarizes the recommended steps to configure a high-availability (HA) Heartbeat cluster for use with SGI InfiniteStorage products. SGI recommends that you create one resource group that contains the set of resources that must be failed over as a unit. The details for these steps are provided in chapters 5–8.

Note: You can add the resources into the Heartbeat cluster information base (CIB) by using the `hb_gui` graphical user interface (GUI) or by writing XML files and using the `cibadmin` command. This guide documents the XML method. For more information about the tools, see "Sources for Detailed Heartbeat Documentation" on page 5.

Do the following:

1. Understand the requirements for the SGI products you want to include in your Heartbeat cluster. See Chapter 3, "Preliminary Requirements for High Availability" on page 9.

2. Select the appropriate YaST patterns as part of the ISSP installation. For example, in order to install or upgrade the software for Heartbeat, DMF, and associated products:

   SGI HA Server
   SGI DMF Server

   For more information, see the *SGI InfiniteStorage Software Platform* release note.

3. Configure and test each of the standard SGI product services before making them highly available. Do this using one host (which will later become a node in the Heartbeat cluster) on which all of the filesystems will be mounted and on which all tape drives and libraries are accessible, known in this guide as the *initial node*. See Chapter 5, "Configuring and Testing the Standard Services" on page 23.

   If you already have stable systems configured, you can skip this step and proceed to step 4.

4. Stop the standard services and unmount all of the filesystems.

5. Establish a Heartbeat cluster that contains the nodes that will be DMF servers. See Chapter 6, "Establishing a Heartbeat Cluster" on page 27.

6. Create an empty resource group to be used as a holding place, to which you will eventually add all of the resources that are required to be failed over together. For example, to create a resource group named `dmfGroup`:

   ```
   initial# cibadmin -o resources -C -X '<group id="dmfGroup"/>'
   ```

7. Constrain the group to start on the initial node:

   ```
   initial# crm_resource -M -t group -r dmfGroup -H initial_node
   ```

8. Configure, add, and test each of the required resources in the following order, as shown in Figure 4-1:

   - Filesystem resources, either:
     - **CXFS resource** using the `cxfs` resource agent
     - **Local XVM resource** using the `lxvm` resource agent and then one or more **filesystem resources** using the community `Filesystem` Open Cluster Framework (OCF) resource agent
   - *(Optional)* **Virtual IP address resource** using the community `IPaddr2` OCF resource agent (required by the OpenVault resource and the NFS server resource)
   - **DMF mounting service resource** using either the SGI `openvault` or SGI `tmf` resource agent
   - **DMF resource** using the SGI `dmf` resource agent
   - **NFS server resource** using the community `nfsserver` Linux Standard Base (LSB) resource agent

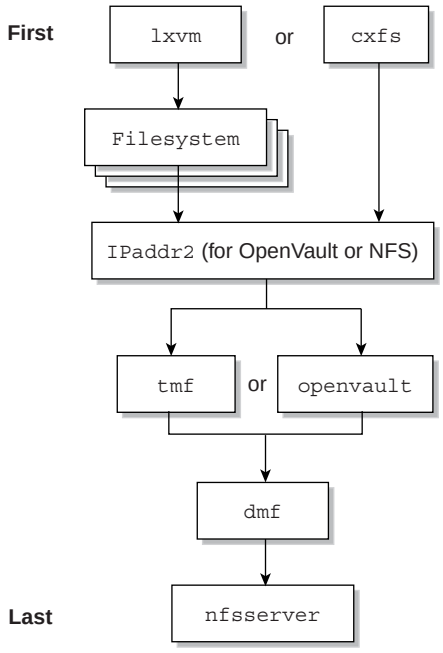

Figure 4-1 Required Resource Configuration Order

Complete the following series of steps for each resource:

a. Write the XML for a resource, surrounding it with the same `<group id="`*groupname*`">` and `</group>` tags. For example, in a file called `local_xvm.xml`, you would include the following (truncated here):

```
<group id="dmfGroup">
<primitive id="local_xvm" class="ocf" provider="sgi" type="lxvm">
... XML tags...
</primitive>
</group>
```

For instructions, see Chapter 7, "Configuring SGI Products for High Availability and Testing" on page 31.

b. Include the resource information by updating the original resource group in the CIB. For example, to include the XML for the local XVM resource:

```
initial# cibadmin -U -o resources -x local_xvm.xml
```

Note: After updating the database, the resource will be immediately actively managed by Heartbeat, with the exception of the OpenVault resource, which at initial configuration will be explicitly withheld from Heartbeat management.

c. Test the resource, according to the instructions in Chapter 7, "Configuring SGI Products for High Availability and Testing" on page 31. Move the resource group from the initial node to an alternate node in the cluster, then back again, testing for appropriate functionality:

```
initial# crm_resource -t group -r dmfGroup -r -M -H alternate_node
(test on the alternate node according to the documentation)
initial# crm_resource -t group -r dmfGroup -r -M -H initial_node
(test on the initial node according to the documentation)
```

Repeat steps 8a through 8c for each required resource. Proceed to the next resource only if the current resource is behaving as expected, as defined by the documentation.

9. Configure system reset. See Chapter 8, "Configuring and Testing STONITH Reset Services" on page 71

Note: System reset using the STONITH facility is required to ensure data integrity.

10. Enable system reset (which is disabled by default):

```
initial# crm_attribute -t crm_config -n stonith-enabled -v true
```

11. Remove the resource location constraints generated by the administrative move commands executed above:

```
initial# crm_resource -t group -r dmfGroup -r -U
```

For details about the above steps, see the procedures in the following chapters:

- Chapter 5, "Configuring and Testing the Standard Services" on page 23
- Chapter 6, "Establishing a Heartbeat Cluster" on page 27
- Chapter 7, "Configuring SGI Products for High Availability and Testing" on page 31
- Chapter 8, "Configuring and Testing STONITH Reset Services" on page 71

Configuring and Testing the Standard Services

You should configure and test all standard services before applying high availability (HA). You should do this on one host (which will later become a node in the Heartbeat cluster) on which all of the filesystems will be mounted and on which all tape drives and libraries are accessible, known in this guide as the *initial node*.

If you already have a stable configuration, you can skip the steps in this chapter and continue to Chapter 6, "Establishing a Heartbeat Cluster" on page 27.

This chapter discusses the following:

Standard CXFS

To configure and test the standard CXFS service before applying high availability, do the following:

1. Configure CXFS using one host as the server-capable administration node (which will later become a node in the Heartbeat cluster) on which all of the filesystems will be mounted, known in this guide as the *initial node*, according to the instructions in the following:
   - "CXFS Requirements" on page 10
   - *CXFS 5 Administration Guide for SGI InfiniteStorage*
   - *CXFS 5 Client-Only Guide for SGI InfiniteStorage*
2. Start CXFS services and CXFS cluster services. For more information, see *CXFS 5 Administration Guide for SGI InfiniteStorage*.

3. Verify that the filesystem in question mounts on all applicable nodes. For example, use the `cxfs_admin` command:

   ```
   initial# cxfs_admin -c status
   ```

Standard Local XVM

Configure, use the `mkfs` command, and mount all of your HA local XVM filesystems on the initial node according to the instructions in the *XVM Volume Manager Administrator's Guide.*

To test the local XVM configuration, ensure that you can create and delete files in each of the mounted filesystems.

Standard OpenVault

Configure OpenVault on the initial node, according to the instructions in the *OpenVault Operator's and Administrator's Guide.* This means that you will use the **actual** hostname as reported by the `hostname`(1) command when using `ov_admin`. Configure all local libraries and tape drives. (Configuration of OpenVault on the other DMF server nodes will not be done until the conversion to HA is performed.)

To test OpenVault, verify that you can perform operational tasks documented in the OpenVault guide, such as mounting and unmounting of cartridges using the `ov_mount` and `ov_unmount` commands.

For example, in an OpenVault configuration with two tape drives (`drive0` and `drive1`) where you have configured a volume named `DMF105` for use by DMF, the following sequence of commands will verify that tape drive `drive0` and the library are working correctly. (Repeat the sequence for `drive1`.)

```
initial# ov_mount -A dmf -V DMF105 -d drive0
Mounted DMF105 on /var/opt/openvault/clients/handles/An96H0uA3xr0
initial# tsmt status
        Controller: SCSI
        Device: SONY: SDZ-130         0202
        Status: 0x20262
        Drive type: Sony SAIT
        Media : READY, writable, at BOT
initial# ov_stat -d | grep DMF105
drive0              drives     true   false  false    inuse     loaded    ready     true         DMF105S1
```

```
initial# ov_unmount -A dmf -V DMF105 -d drive0
Unmounted DMF105
initial# exit
```

Standard TMF

Configure TMF on the initial node according to the instructions in the *TMF Administrator's Guide* and run the following on the initial node:

```
initial# chkconfig -s tmf on
```

Note: In the `tmf.config` file, drives in drive groups managed by Heartbeat should have `access` configured as `EXCLUSIVE` and should have `status` configured as `DOWN` when TMF starts. Loaders in the `tmf.config` file should have status configured as `UP` when TMF starts.

To test TMF, do the following:

1. Use `tmstat` to verify that all the tape drives have a status of `idle` or `assn`:

   ```
   initial# tmstat
   ```

2. Use `tmmls` to verify that all of the loaders have a status of `UP`:

   ```
   initial# tmmls
   ```

Standard DMF

Configure DMF according to the instructions in the *DMF Administrator's Guide for SGI InfiniteStorage*.

To test DMF, do the following:

1. Migrate a few test files:

   ```
   initial# dmput -r files_to_test
   ```

2. Force tapes to be immediately written:

   ```
   initial# dmdidle
   ```

   Wait a bit to allow time for the tape to be written and unmounted.

3. Verify that the tapes are mounted and written successfully.
4. Verify that the tapes can be read and the data can be retrieved:

```
initial# dmget files_to_test
```

Standard NFS Serving

Set up the NFS exports in the /etc/exports file on the initial node as you would normally.

To test the NFS service, do the following:

1. Run the following command on the initial node to verify the NFS filesystems are exported:

```
initial# exportfs -v
/work.mynode1   <world>(rw,wdelay,root_squash,no_subtree_check)
/work.mynode2   <world>(rw,wdelay,root_squash,no_subtree_check)
/work.mynode3   <world>(rw,wdelay,root_squash,no_subtree_check)
/work.mynode4   <world>(rw,wdelay,root_squash,no_subtree_check)
/mirrors        <world>(ro,wdelay,root_squash,no_subtree_check)
/               <world>(ro,wdelay,root_squash,no_subtree_check)
```

2. Mount the filesystems on a node that will not be a DMF server (third):

```
third# mount initial:/nfsexportedfilesystem /mnt/test
```

3. Read and write to the NFS-mounted filesystems:

```
third# echo "test data for a test file" > /mnt/test/testFile1A
third# cat /mnt/test/testFile1A
test data for a test file
```

Establishing a Heartbeat Cluster

This chapter presents an overview of the steps required to establish a Heartbeat cluster. For more details, see the information referred to in "Sources for Detailed Heartbeat Documentation" on page 5.

1. Ensure that you have unmounted all of the filesystems and stopped the standard services configured in Chapter 5, "Configuring and Testing the Standard Services".
2. On the initial node:
   a. Copy the `authkeys` and `ha.cf` files from the `/usr/share/doc/heartbeat-`*VERSION* directory to the `/etc/ha.d` directory. For example:

      ```
      initial# cd /usr/share/doc/heartbeat-VERSION
      initial# cp ./authkeys  ./ha.cf  /etc/ha.d/
      ```

      Note: SGI requires that you use Heartbeat version 2, for which the `haresources` file is not required.

   b. Change the permissions on the `/etc/ha.d/authkeys` file to allow read and write permission for `root` only. For example:

      ```
      initial# chmod 600 /etc/ha.d/authkeys
      ```

   c. In the `/etc/ha.d/authkeys` file, uncomment the following lines:

      ```
      auth 1
      1 crc
      ```

      You could also use the SHA1 and MD5 authentication methods, which require further configuration. For more information, see "Sources for Detailed Heartbeat Documentation" on page 5.

   d. In the `/etc/ha.d/ha.cf` file, do the following:

      - Comment out the `logfacility` directive.
      - Assign a value to the `bcast`, `mcast`, or `ucast` directive indicating the interface that will be used to send heartbeats.

For example, using `eth1`for heartbeats assuming the IP for `node-0` is `128.162.244.220` and the IP for `node-1` is `128.162.244.221`:

```
ucast eth1 128.162.244.220
ucast eth1 128.162.244.221
```

Note: When configuring the `ucast` directive in the `ha.cf` file on the initial node, make a `ucast` entry for each node (including the initial node) in the cluster. The `mcast` directive requires additional configuration information. For more information, see "Sources for Detailed Heartbeat Documentation" on page 5.

- Add the following directives:

```
node nodename      (one for each cluster node)
use_logd yes
crm yes
```

e. Configure Heartbeat so that it will start upon reboot:

```
initial# chkconfig heartbeat on
```

f. Distribute the configuration by using the following command (you must know the `root` password for every node added to the `ha.cf` file):

- Altix:

```
initial# /usr/lib/heartbeat/ha_propagate
```

- Altix XE:

```
initial# /usr/lib64/heartbeat/ha_propagate
```

Example 6-1 shows portions of a corrected `/etc/ha.d/ha.cf` file. Highlighted lines have been modified from the original in the `/usr/share/doc/heartbeat-`*VERSION* directory.

Example 6-1 Corrected `ha.cf` File

```
#       Facility to use for syslog()/logger
#
#logfacility    local0
#
...
#       What interfaces to broadcast heartbeats over?
#
ucast eth1 node-0
ucast eth1 node-1
...
#       Tell what machines are in the cluster
#       node    nodename ...    -- must match uname -n
#node   node-2
#node   node-3
node node-0
node node-1
...
# use_logd yes/no
use_logd yes
...
crm yes
```

3. On all nodes:

   a. *(Optional but recommended)* Add directives to `/etc/sysctl.conf` for the `kernel.core_pattern` and `kernel.core_uses_pid` variables. For example:

      ```
      kernel.core_pattern = core.%p.%t
      kernel.core_uses_pid = 1
      ```

      Changes made to `/etc/sysctl.conf` will take effect on the next boot and will be persistent. To make the settings effective immediately for the current session only, enter the following:

      ```
      # echo 1 > /proc/sys/kernel/core_uses_pid
      # echo "core.%p.%t" > /proc/sys/kernel/core_pattern
      ```

   b. Set the password for the `hacluster` user if you intend to use the `hb_gui` tool.

c. Start Heartbeat:

```
# /etc/init.d/heartbeat start
```

4. Test the base Heartbeat cluster by running the following command, waiting to see both nodes come online (could take a few minutes):

```
# crm_mon -ri1
```

Configuring SGI Products for High Availability and Testing

This chapter tells you how to configure SGI products for high availability (HA), create the HA resources using the XML method, and how to test them. For a complete XML example, see Appendix A, "Complete XML Examples" on page 87.

Note: The XML examples in this chapter show values that are site-specific in bold; these are the values you will want to change. You should not change other values unless you have good reason to do so and are aware of the implications. Required values are noted.

You must configure and test resources in the following in order, skipping products that do not apply to your site:

1. Filesystems. Either:
   - CXFS:
     - "Configuring CXFS for HA " on page 32
     - "Testing the `cxfs` Resource" on page 34
   - Local XVM and one or more filesystems:
     - "Configuring Local XVM for HA" on page 35
     - "Testing the `lxvm` Resource" on page 37
     - "Configuring Filesystems for HA" on page 38
     - "Testing `Filesystem` Resources" on page 43
2. A virtual IP address (for OpenVault or NFS):
   - "Configuring a Virtual IP Address for HA" on page 44
   - "Testing the `IPaddr2` Resource" on page 46
3. A mounting service, either OpenVault or TMF:
   - "Configuring OpenVault for HA" on page 47

Configuring CXFS for HA

To configure CXFS for HA, do the following:

1. Write an XML file for the CXFS resource. Example 7-1 shows a CXFS resource instance in a file named `cxfs.xml`.

Example 7-1 CXFS Resource XML

```
<group id="dmfGroup">
<primitive id="cxfs_resource" class="ocf" provider="sgi" type="cxfs">
    <instance_attributes>
        <attributes>
            <nvpair name="volnames" value="cxfs_volume1,cxfs_volume2"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="600s" on_fail="restart" prereq="fencing"/>
        <op name="stop" timeout="600s" on_fail="fence"/>
        <op name="monitor" timeout="20s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
```

```
        </attributes>
    </meta_attributes>
</primitive>
</group>
```

The tag elements and their attributes are as follows:

- `group id` specifies the name of the resource group to which this resource will belong, in this case `dmfGroup`.
- `primitive id` is the name of the resource in this case `cxfs_resource`. The element's attributes must be of class `ocf`, provider `sgi`, and type `cxfs`.
- `volnames` is the name of a resource attribute. The value specified for `volumes` is a comma-separated list of CXFS filesystems to be managed, in this case volumes `cxfs_volume1` and `cxfs_volume2`.

Note: The CXFS resource agent supports a single resource instance per CXFS cluster so that the metadata server for all of the filesystems managed by Heartbeat will be co-located on the same server-capable administration node.

- `start` is the name of the operation that initiates or gains control of the resource. It will timeout after 600 seconds. It requires that fencing is configured and active in order to start the resource. Using system reset as a fencing method is required in order to preserve data integrity. If the `start` operation fails, it will attempt to restart the resource.
- `stop` is the name of the operation that terminates or gives up control of the resource. It will timeout after 600 seconds. If the `stop` operation fails, it will attempt to fence the node on which the failure occurred. The `stop` fail policy must be set to `fence` and a STONITH facility must be configured according to the requirements for your site.
- `monitor` is the name of the operation that determines if the resource is operating correctly. Each `monitor` operation will timeout after 20 seconds. If the `monitor` operation fails, it will attempt to restart the resource.
- `resource_stickiness` specifies a weight for the preference to keep this resource on the node on which it is currently running. A positive value specifies a preference for the resource to remain on the node on which it is currently running. This preference may only be overridden if the node becomes ineligible to run the resource (if the node goes into standby mode) or

if there is a `start`, `monitor`, or `stop` failure for this resource or another resource in the same resource group.

- `resource_failure_stickiness` specifies a weight for the preference to move this resource to a new node based on the number of `start`, `monitor`, or `stop` failures that this resource has experienced. The value of `-INFINITY` specifies that if this resource experiences a failure, this resource and all members of its resource group will be restarted on a different eligible node.

2. Include the CXFS resource in the CIB. For example:

```
initial# cibadmin -U -o resources -x cxfs.xml
```

Testing the `cxfs` Resource

To test the `cxfs` resource in a resource group named `dmfGroup`, do the following:

1. Verify that CXFS is working on the initial node. For example, use the `df` command to verify that all of the CXFS filesystems are mounted and accessible, and use the `cxfs_admin` command `show server` to display the current metadata server for the filesystems:

```
initial# df -lh
initial# /usr/cluster/bin/cxfs_admin -c "show server"
```

2. Move the resource group containing the `cxfs` resource to an alternate node :

```
initial# crm_resource -t group -r dmfGroup -M -H alternate_node
```

3. Verify that CXFS is working on the alternate node:

```
alternate# df -lh
alternate# /usr/cluster/bin/cxfs_admin -c "show server"
```

4. Move the resource group containing the `cxfs` resource back to the initial node:

```
initial# crm_resource -t group -r dmfGroup -M -H initial_node
```

5. Verify that CXFS is working again on the initial node:

```
initial# df -lh
initial# /usr/cluster/bin/cxfs_admin -c "show server"
```

6. Remove the resource location constraints generated by the administrative move commands executed above:

```
initial# crm_resource -t group -r dmfGroup -U
```

Configuring Local XVM for HA

To configure local XVM for HA, do the following:

1. Write an XML file for the local XVM resource. Example 7-2 shows a local XVM resource instance in a file named `local_xvm.xml`.

Example 7-2 Local XVM Resource XML

```
<group id="dmfGroup">
<primitive id="local_xvm" class="ocf" provider="sgi" type="lxvm">
    <instance_attributes>
        <attributes>
            <nvpair name="volnames" value="openvault,home,journals,spool,move,tmp,diskmsp,dmfusr1,dmfusr3"/>
            <nvpair name="physvols" value="myCluster,myClusterStripe1,myClusterStripe2"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="60s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="60s" on_fail="fence"/>
        <op name="monitor" interval="20s" timeout="40s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
</group>
```

The tag elements and their attributes are as follows:

- `group id` specifies the name of the resource group to which this resource will belong, in this case `dmfGroup`.
- `primitive id` specifies the name of the resource, in this case `local_xvm`. The element's attributes must be of class `ocf`, provider `sgi`, and type `lxvm`.

- volnames is a comma-separated list of local XVM volume names to monitor, in this case openvault, home, journals, spool, move, tmp, diskmsp, dmfusr1, and dmfusr3.
- physvols is a comma-separated list of the physical volumes (*physvols*) for the resource agent to steal, in this case myCluster, myClusterStripe1, and myClusterStripe2. physvols must contain all of the physical volumes for every logical volume listed in volnames.

Note: All physical disks that belong to a logical volume in an HA cluster must be completely dedicated to that logical volume and no other.

- start is the name of the operation that initiates the resource. In this case, it will timeout after 60 seconds. (While a 60-second timeout should be sufficient in most cases, some sites with large disk configurations may need to adjust this timeout value.) It requires that fencing is configured and active in order to start the resource. Using system reset as a fencing method is required in order to preserve data integrity. If the start operation fails, it will attempt to restart the resource.
- stop is the name of the operation that terminates or gives up control of the resource. In this case, it will timeout after 60 seconds. (You should use the same timeout value that you used for the start operation.) If the stop operation fails, it will attempt to fence the node on which the failure occurred. The stop fail policy must be set to fence and a STONITH facility must be configured according to the requirements for your site.
- monitor is the name of the operation that determines if the resource is operating correctly. In this case, the resource will be monitored at an interval of 20 seconds. Each monitor operation will timeout after 40 seconds. If the monitor operation fails, it will attempt to restart the resource.
- resource_stickiness specifies a weight for the preference to keep this resource on the node on which it is currently running. A positive value specifies a preference for the resource to remain on the node on which it is currently running. This preference may only be overridden if the node becomes ineligible to run the resource (if the node goes into standby mode) or if there is a start, monitor, or stop failure for this resource or another resource in the same resource group.
- resource_failure_stickiness specifies a weight for the preference to move this resource to a new node based on the number of start, monitor,

or stop failures that this resource has experienced. The value of -INFINITY specifies that if this resource experiences a failure, this resource and all members of its resource group will be restarted on a different eligible node.

2. Include the local XVM resource in the CIB. For example:

```
initial# cibadmin -U -o resources -x local_xvm.xml
```

Testing the lxvm Resource

To test the lxvm resource in a resource group named dmfGroup, do the following:

1. Move the resource group containing the lxvm resource from the initial node to an alternate node:

```
initial# crm_resource -t group -r dmfGroup -M -H alternate_node
```

Note: If the timeout is too short for a start operation, the crm_mon and crm_verify -LV output and the /var/log/messages file will have an entry that says refers to the action being "Timed Out". For example (line breaks shown here for readability):

```
initial# crm_mon -1 | grep Timed
    lxvm_resource_start_0 (node=node-0, call=222, rc=-2): Timed Out

initial# crm_verify -LV 2>&1 | grep Timed
crm_verify[147386]: 2008/07/23_14:36:34 WARN: unpack_rsc_op:
  Processing failed op lxvm_resource_start_0
  on node-0: Timed Out
```

---

2. Look at /dev/lxvm on the alternate node to verify the change. Each of the volumes listed in the volnames attribute should appear within the /dev/lxvm directory. (There may be additional volumes within the directory.)

3. Move the resource group containing the lxvm resource back to the initial node:

```
initial# crm_resource -t group -r dmfGroup -M -H initial_node
```

4. Look at /dev/lxvm on the initial node to verify the change as described in step 2.

5. Remove the resource location constraints generated by the administrative move commands executed above:

```
initial# crm_resource -t group -r dmfGroup -U
```

Configuring Filesystems for HA

In this release, SGI supports the following types of filesystems:

- DMF-managed user filesystems

  Note: The optional `MOVE_FS` DMF administrative filesystem and all DMF-managed user filesystems require `dmi` and `mtpt` mount options to be specified when configuring their resources.

- DMF administrative filesystems specified by the following parameters in the DMF configuration file:

  `HOME_DIR`
  `JOURNAL_DIR`
  `SPOOL_DIR`
  `TMP_DIR`
  `MOVE_FS`
  `CACHE_DIR` for any Library Servers
  `STORE_DIRECTORY` for any DCMs and disk MSPs using local disk storage

- *(Optional)* OpenVault server filesystem
- *(Optional)* Any additional HA filesystems that are not managed by DMF; for example, other NFS-exported filesystems that are not under DMF control

All of the above filesystems should be configured as locally mounted XFS filesystems using the following resources:

- Local XVM resource
- Community-provided `Filesystem` resource

For additional information, see:

- http://linux-ha.org/OCFResourceAgent
- http://linux-ha.org/OCF

This section discusses the following:

- "DMF-Managed User `Filesystem` Resource" on page 39
- "DMF Administrative `Filesystem` Resource" on page 41

• "Dedicated OpenVault Server `Filesystem` Resource *(Optional)*" on page 42

DMF-Managed User `Filesystem` Resource

To configure a DMF-managed user filesytem for HA, do the following:

1. Edit `/etc/fstab` and remove all of the filesystems that you will manage with Heartbeat.
2. Write an XML file for the filesystem resource. Example 7-3 shows a DMF-managed user filesystem resource instance in a file named `dmfusr1FS.xml`.

Example 7-3 DMF-Managed User `Filesystem` Resource XML

```
<group id="dmfGroup">
<primitive id="dmfusr1FS" class="ocf" provider="heartbeat" type="Filesystem">
    <instance_attributes>
        <attributes>
            <nvpair name="device" value="/dev/lxvm/dmfusr1"/>
            <nvpair name="directory" value="/dmfusr1"/>
            <nvpair name="fstype" value="xfs"/>
            <nvpair name="options" value="dmi,mtpt=/dmfusr1"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="15s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="10s" on_fail="fence"/>
        <op name="monitor" interval="10s" timeout="10s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
</group>
```

The tag elements and their attributes are as follows:

- `group id` specifies the name of the resource group to which this resource will belong, in this case `dmfGroup`.

- `primitive id` specifies the name of the resource, in this case `dmfusr1FS`. The element's attributes must be of class `ocf`, provider `heartbeat`, and type `Filesystem`.
- `device` is the `/dev/lxvm` volume name of the DMF spool filesystem device, `/dev/lxvm/dmfusr1` in this case.
- `directory` is the mount point for the DMF spool filesystem, `/dmfusr1` in this case.
- `options` lists the mount options. The value of the `mtpt` mount option must match the value used for the `directory` attribute, `/dmfusr1` in this case.
- `fstype` is the filesystem type (must be `xfs`).
- `start` is the name of the operation that initiates the resource. It will timeout after 15 seconds. It requires that fencing is configured and active in order to start the resource. Using system reset as a fencing method is required in order to preserve data integrity. If the `start` operation fails, it will attempt to restart the resource.
- `stop` is the name of the operation that terminates or gives up control of the resource. It will timeout after 10 seconds. If the stop operation fails, it will attempt to fence the node on which the failure occurred. The `stop` fail policy must be set to `fence` and a STONITH facility must be configured according to the requirements for your site.
- `monitor` is the name of the operation that determines if the resource is operating correctly. The resource will be monitored at an interval of 10 seconds. Each `monitor` operation will timeout after 10 seconds. If the monitor operation fails, it will attempt to restart the resource.
- `resource_stickiness` specifies a weight for the preference to keep this resource on the node on which it is currently running. A positive value specifies a preference for the resource to remain on the node on which it is currently running. This preference may only be overridden if the node becomes ineligible to run the resource (if the node goes into standby mode) or if there is a `start`, `monitor`, or `stop` failure for this resource or another resource in the same resource group.
- `resource_failure_stickiness` specifies a weight for the preference to move this resource to a new node based on the number of `start`, `monitor`, or `stop` failures that this resource has experienced. The value of `-INFINITY`

specifies that if this resource experiences a failure, this resource and all members of its resource group will be restarted on a different eligible node.

3. Include the resource in the CIB. For example:

```
initial# cibadmin -U -o resources -x dmfusr1FS.xml
```

DMF Administrative `Filesystem` Resource

To configure a DMF administrative filesytem for HA, do the following:

1. Ensure that you have edited `/etc/fstab` and removed all of the filesystems that you will manage with Heartbeat.
2. Write an XML file for the filesystem resource. Example 7-4 shows a DMF administrative filesystem resource instance in a file named `spoolFS.xml`.

Example 7-4 DMF Administrative `Filesystem` Resource XML

```
<group id="dmfGroup">
<primitive id="spoolFS" class="ocf" provider="heartbeat" type="Filesystem">
    <instance_attributes>
        <attributes>
            <nvpair name="device" value="/dev/lxvm/spool"/>
            <nvpair name="directory" value="/dmf/spool"/>
            <nvpair name="fstype" value="xfs"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="15s" prereq="fencing" on_fail="restart" />
        <op name="stop" timeout="10s" on_fail="fence"/>
        <op name="monitor" interval="10s" timeout="10s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
</group>
```

Note: You do not need the `options` attribute for any DMF administrator filesystem except the optional `MOVE_FS` filesystem, which that requires `dmi` and `mtpt` mount options to be specified.

3. Include the resource in the CIB. For example:

```
initial# cibadmin -U -o resources -x spoolFS.xml
```

Dedicated OpenVault Server `Filesystem` Resource *(Optional)*

If you choose to have a dedicated filesystem for the OpenVault `serverdir` directory for HA, do the following:

1. Write an XML file for the filesystem resource. Example 7-5 shows the XML for an optional separate HA filesystem for the OpenVault `serverdir` directory in a file named `openvaultFS.xml`.

Example 7-5 OpenVault `serverdir` `Filesystem` Resource XML

```
<group id="dmfGroup">
<primitive id="openvaultFS" class="ocf" provider="heartbeat" type="Filesystem">
    <instance_attributes>
        <attributes>
            <nvpair name="device" value="/dev/lxvm/openvault"/>
            <nvpair name="fstype" value="xfs"/>
            <nvpair name="directory" value="/dmf/openvault"/>
        </attributes>
    </instance_attributes>
    <operations>
        op name="start" timeout="15s" prereq="fencing" on_fail="restart" />
        op name="stop" timeout="10s" on_fail="fence"/>
        op name="monitor" interval="10s" timeout="10s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
</group>
```

2. Include the resource in the CIB. For example:

```
initial# cibadmin -U -o resources -x openvaultFS.xml
```

Testing `Filesystem` Resources

To test the filesystem resources for a DMF resource group named `dmfGroup`, do the following:

1. Move the resource group containing all of the `Filesystem` resources from the initial node to an alternate node:

```
initial# crm_resource -t group -r dmfGroup -M -H alternate_node
```

2. Verify that the filesystems are correctly mounted on the alternate node only:

   - On the alternate node, check the mount table and verify that the filesystems are mounted and have the correct mount options. Use the `ls` and `df -lh` commands on the mount point to verify that the filesystem is functional.
   - On the initial node, check the mount table and verify that none of the filesystems are mounted.

3. Move the the resource group containing all of the `Filesystem` resources back to the initial node:

```
initial# crm_resource -t group -r dmfGroup -M -H initial_node
```

4. Verify that the filesystems are correctly mounted on the initial node only:

   - On the initial node, check the mount table and verify that the filesystems are mounted and have the correct mount options. Use the `ls` and `df -lh` commands on the mount point to verify that the filesystem is functional.
   - On the alternate node, check the mount table and verify that none of the filesystems are mounted.

5. Remove the resource location constraints generated by the administrative move commands executed above:

```
initial# crm_resource -t group -r dmfGroup -U
```

Configuring a Virtual IP Address for HA

To create the virtual IP address for HA, do the following:

1. Write an XML file for the virtual IP address resource. Example 7-6 shows a virtual IP address resource instance in a file named `VirtualIP.xml`.

Example 7-6 Virtual IP Address Resource XML

```
<group id="dmfGroup">
<primitive id="VirtualIP" class="ocf" provider="heartbeat" type="IPaddr2" >
    <instance_attributes>
        <attributes>
            <nvpair name="ip" value="128.162.244.240"/>
            <nvpair name="nic" value="eth2"/>
            <nvpair name="cidr_netmask" value="255.255.255.0"/>
            <nvpair name="broadcast" value="128.162.244.255"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="10s" prereq="fencing" on_fail="restart" />
        <op name="stop" timeout="10s" on_fail="fence"/>
        <op name="monitor" interval="10s" timeout="10s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
</group>
```

The tag elements and their attributes are as follows:

- `group id` specifies the name of the resource group to which this resource will belong, in this case `dmfGroup`.
- `primitive id` specifies the name of the resource, in this case `VirtualIP`. The element's attributes must be of class `ocf`, provider `heartbeat`, and type `IPaddr2`.
- `ip` is the virtual IP address, in this case `128.162.244.240`.

- `nic` is the network interface card that will service the virtual IP address, in this case `eth2`.
- `cidr_netmask` is the network mask.
- `broadcast` is the broadcast IP address, in this case `128.162.244.255`.
- `start` is the name of the operation that initiates the resource. It will timeout after 10 seconds. It requires that fencing is configured and active in order to start the resource. Using system reset as a fencing method is required in order to preserve data integrity. If the `start` operation fails, it will attempt to restart the resource.
- `stop` is the name of the operation that terminates or gives up control of the resource. It will timeout after 10 seconds. If the stop operation fails, it will attempt to fence the node on which the failure occurred. The `stop` fail policy must be set to `fence` and a STONITH facility must be configured according to the requirements for your site.
- `monitor` is the name of the operation that determines if the resource is operating correctly. The resource will be monitored at an interval of 10 seconds. Each `monitor` operation will timeout after 10 seconds. If the monitor operation fails, it will attempt to restart the resource.
- `resource_stickiness` specifies a weight for the preference to keep this resource on the node on which it is currently running. A positive value specifies a preference for the resource to remain on the node on which it is currently running. This preference may only be overridden if the node becomes ineligible to run the resource (if the node goes into standby mode) or if there is a `start`, `monitor`, or `stop` failure for this resource or another resource in the same resource group.
- `resource_failure_stickiness` specifies a weight for the preference to move this resource to a new node based on the number of `start`, `monitor`, or `stop` failures that this resource has experienced. The value of `-INFINITY` specifies that if this resource experiences a failure, this resource and all members of its resource group will be restarted on a different eligible node.

2. Include the resource in the CIB. For example:

```
initial# cibadmin -U -o resources -x VirtualIP.xml
```

For additional information, see:

http://linux-ha.org/HeartbeatResourceAgent/IPaddr2

Testing the IPaddr2 Resource

To test the IPaddr2 resource, do the following:

1. Verify that the IP address is configured correctly on the initial node:

```
initial# ip -o addr show | grep 128.162.244.240
4: eth2    inet 128.162.244.240/24 brd 128.162.244.255 scope global secondary eth2
```

2. Verify that the alternate node does not accept the IP address packets by running the following command on the alternate node (the output should be 0):

```
alternate# ip -o addr show | grep -c 128.162.244.240
0
```

3. Connect to the virtual address using ssh or telnet and verify that the IP address is being served by the correct system.

   For example, for the IP address 128.162.244.240 and the machine named node1:

```
# ssh root@128.162.244.240
Last login: Mon Jul 14 10:34:58 2008 from mynode.mycompany.com
# uname -n
node1
```

4. Move the resource group containing the IPaddr2 resource from the initial node to an alternate node:

```
initial# crm_resource -t group -r dmfGroup -M -H alternate_node
```

5. Verify that the IP address is configured correctly on the alternate node:

```
alternate# ip -o addr show | grep 128.162.244.240
4: eth2    inet 128.162.244.240/24 brd 128.162.244.255 scope global secondary eth2
```

6. Verify that the initial node does not accept the IP address packets by running the following command on the initial node (the output should be 0):

```
initial# ip -o addr show | grep -c 128.162.244.240
0
```

7. Connect to the virtual address using `ssh` or `telnet` and verify that the IP address is being served by the correct system. For example, for the IP address `128.162.244.240` and the machine named `node2`:

```
# ssh root@128.162.244.240
Last login: Mon Jul 14 10:34:58 2008 from mynode.mycompany.com
# uname -n
node2
```

8. Move the resource group containing the `IPaddr2` resource back to the initial node:

```
initial# crm_resource -t group -r dmfGroup -M -H initial_node
```

9. Test again as in steps 1—3 above.

10. Remove the resource location constraints generated by the administrative move commands executed above:

```
initial# crm_resource -t group -r dmfGroup -U
```

Configuring OpenVault for HA

Do the following:

1. Ensure that all the resources within the resource group are moved back to the the initial node (if not already there).

2. Write the XML for the OpenVault resource and initially configure it with Heartbeat management disabled (set the parameter `is_managed` to `false`) so that the conversion to an HA configuration can be completed before Heartbeat begins managing it. (In step 7 below, parameter `is_managed` will be set to `true`, causing Heartbeat to then begin management of the OpenVault resource.)

   Example 7-7 shows an OpenVault resource instance in a file named `OpenVault.xml`.

Example 7-7 OpenVault Resource XML

```
<group id="dmfGroup">
<primitive id="OpenVault" class="ocf" provider="sgi" type="openvault">
    <instance_attributes>
       <attributes>
            <nvpair name="virtualhost" value="myIP.company.com"/>
            <nvpair name="serverdir" value="/dmf/openvault/server"/>
            <nvpair name="is_managed" value="false"/>
       </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="300s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="30s" on_fail="fence"/>
        <op name="monitor" interval="60s" timeout="30s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
</group>
```

The tag elements and their attributes are as follows:

- `group id` specifies the name of the resource group to which this resource will belong, in this case `dmfGroup`.
- `primitive id` specifies the name of the resource, in this case `OpenVault`. The element's attributes must be of class `ocf`, provider `sgi`, and type `openvault`. There can only be one resource of type `openvault` within the Heartbeat cluster.
- `virtualhost` is a hostname that will resolve as defined in `/etc/nsswitch.conf` to an IP address managed via an `IPaddr2` resource within the same resource group as the OpenVault resource (see "Configuring a Virtual IP Address for HA" on page 44). In this example, the DNS name is `myIP.company.com`.
- `serverdir` is a path on a mountable XFS filesystem that is being managed by a `Filesystem` resource in the same resource group as the `openvault`

resource. The filesystem could be one dedicated for OpenVault use (such as `/dmf/openvault`) or it could be a Heartbeat-managed filesystem in the same resource group that has sufficient space (such as `/dmf/home/`). As part of the conversion to HA, OpenVault will create this directory and move its database and logs into the directory, so the directory must not already exist or OpenVault will fail.

- `is_managed` specifies whether Heartbeat will immediately begin managing the resource as soon as it is added to the CIB. Unlike other resources, this value must be set to `false` for OpenVault during the initial configuration process so that the conversion to an HA configuration can be completed before before Heartbeat begins managing it.
- `start` is the name of the operation that initiates the resource. In this case, it will timeout after 300 seconds. It requires that fencing is configured and active in order to start the resource. Using system reset as a fencing method is required in order to preserve data integrity. If the `start` operation fails, it will attempt to restart the resource.
- `stop` is the name of the operation that terminates or gives up control of the resource. In this case, it will timeout after 30 seconds. If the stop operation fails, it will attempt to fence the node on which the failure occurred. The `stop` fail policy must be set to `fence` and a STONITH facility must be configured according to the requirements for your site.
- `monitor` is the name of the operation that determines if the resource is operating correctly. In this case, the resource will be monitored at an interval of 60 seconds. Each `monitor` operation will timeout after 30 seconds. If the monitor operation fails, it will attempt to restart the resource.
- `resource_stickiness` specifies a weight for the preference to keep this resource on the node on which it is currently running. A positive value specifies a preference for the resource to remain on the node on which it is currently running. This preference may only be overridden if the node becomes ineligible to run the resource (if the node goes into standby mode) or if there is a `start`, `monitor`, or `stop` failure for this resource or another resource in the same resource group.
- `resource_failure_stickiness` specifies a weight for the preference to move this resource to a new node based on the number of `start`, `monitor`, or `stop` failures that this resource has experienced. The value of `-INFINITY` specifies that if this resource experiences a failure, this resource and all members of its resource group will be restarted on a different eligible node.

3. Include the resource in the CIB. For example:

```
initial# cibadmin -U -o resources -x OpenVault.xml
```

4. Run ov_admin on the initial node:

```
initial# ov_admin
...
```

When asked for the server hostname, specify the virtual hostname (the virtualhost value). ov_admin will automatically convert the OpenVault configuration to a Heartbeat configuration by doing the following:

a. Stopping the server (if it is running)

b. Creating the directory specified by serverdir

c. Moving the OpenVault database and logs into the directory specified by serverdir

d. Making the host specified by virtualhost be the address used by the OpenVault server and all DCPs and LCPs on the initial host.

5. Configure the remaining DMF nodes by using the following steps, repeating the entire sequence for each alternate node before moving to step 6. Whenever ov_admin asks for the server hostname, use the virtual hostname, both on the initial and on the alternate nodes.

Complete the following series of steps for each alternate node:

a. On the initial node:

To allow an alternate node to access the OpenVault server, run ov_admin and answer yes when prompted to start the server. Then select the following menus, answering the questions when prompted:

```
initial# ov_admin
...
Would you like to start the server now? [No] yes
...
23 - Manage Applications and Admin CLI Tools
        7 - Activate a Host to use the Admin CLI Tools
...
```

b. On the alternate node:

   i. Use ov_admin to enable the node to issue administrative commands by selecting:

   ```
   alternate# ov_admin
   ...
   31 - Enable Administrative Commands
   ```

   ii. Configure drives by selecting:

   ```
   2 - Manage DCPs for locally attached Drives
   ```

   You must configure on the alternate node at least one DCP for each drive that is already configured on the initial node; use the same drive name, but a different DCP name.

   iii. Configure libraries by selecting:

   ```
   1 - Manage LCPs for locally attached Libraries
   ```

   You must configure on the alternate node at least one LCP for each library that is already configured on the initial node; use the same library name, but a different LCP name.

   All DCPs and LCPs have now been configured and started on the alternate node, but the server has not yet been configured to allow them to connect. This will be accomplished in step c.

c. On the initial node:

   i. Use ov_admin to enable remote DCPs on the alternate node by selecting:

   ```
   initial# ov_admin
   ...
   22 - Manage remote Drives and DCPs
   ```

   You must enable each remote DCP using the same drive and DCP names that you used on the alternate node.

   ii. Enable the remote libraries on the alternate node by selecting:

   ```
   21 - Manage remote Libraries and LCPs
   ```

   You must enable each remote LCP using the same library and LCP names that you used on the alternate node.

Now that server configuration is complete, the DCPs and the LCPs on the alternate node will shortly discover that they are able to connect to the server.

d. On the alternate node:

   i. Verify that the DCPs are running successfully. For example, the following output shows under `DCPHost` and `DCPStateSoft` columns that the DCP is running and connected to the OpenVault server (`ready`) on the active HA node (`initial`) and running in standby mode (`disconnected`) on the standby HA node (`alternate`):

```
alternate# ov_dumptable -c DriveName,DCPName,DCPHost,DCPStateSoft DCP
DriveName  DCPName          DCPHost DCPStateSoft
9940B_25a1 9940B_25a1@initial initial    ready
9940B_b7ba 9940B_b7ba@initial initial    ready
9940B_93c8 9940B_93c8@initial initial    ready
LTO2_682f  LTO2_682f@initial  initial    ready
LTO2_6832  LTO2_6832@initial  initial    ready
LTO2_6835  LTO2_6835@initial  initial    ready
LTO2_6838  LTO2_6838@initial  initial    ready
9940B_25a1 9940B_25a1@alternate alternate    disconnected
9940B_93c8 9940B_93c8@alternate alternate    disconnected
9940B_b7ba 9940B_b7ba@alternate alternate    disconnected
LTO2_682f  LTO2_682f@alternate  alternate    disconnected
LTO2_6832  LTO2_6832@alternate  alternate    disconnected
LTO2_6838  LTO2_6838@alternate  alternate    disconnected
LTO2_6835  LTO2_6835@alternate  alternate    disconnected
```

Note: It may take a minute or two for the DCPs to notice that they are able to connect to the server and activate themselves. All of the alternate DCPs should transition to `disconnected` state, meaning that they have successfully contacted the server. Do not proceed until they all transition to `disconnected`. A state of `inactive` means that the DCP has not contacted the server, so if the state remains `inactive` for more than a couple of minutes, the DCP may be having problems connecting to the server.

   ii. Verify that the LCPs are running. For example, the following output shows under `LCPHost` and `LCPStateSoft` columns that the LCP is running and connected to the OpenVault server (`ready`) on the active HA

node (initial) and running in standby mode (disconnected) on the standby HA node (alternate):

```
alternate# ov_dumptable -c LibraryName,LCPName,LCPHost,LCPStateSoft LCP
LibraryName LCPName      LCPHost LCPStateSoft
SL500-2     SL500-2@initial initial    ready
L700A       L700A@initial   initial    ready
SL500-2     SL500-2@alternate alternate    disconnected
L700A       L700A@alternate   alternate    disconnected
```

Note: It may take a minute or two for the LCPs to notice that they are able to connect to the server and activate themselves. All of the alternate LCPs should transition to disconnected state, meaning that they have successfully contacted the server. Do not proceed until they all transition to disconnected. A state of inactive means that the LCP has not contacted the server, so if the state remains inactive for more than a couple of minutes, the LCP may be having problems connecting to the server.

iii. Stop all DCPs and LCPs on the alternate server:

```
alternate# ov_stop
```

iv. Disable OpenVault from being started automatically during the boot process:

```
alternate# chkconfig -s openvault off
```

6. On the initial node, stop the OpenVault server and any DCPs and LCPs and turn OpenVault off on the initial node upon reboot:

```
initial# ov_stop
initial# chkconfig -s openvault off
```

7. Update the contents of the OpenVault resource so that is_managed is set to true in the CIB, thus allowing Heartbeat to manage the resource immediately:

```
initial# crm_resource -r OpenVault_resource_name -p is_managed -v true
```

8. *(Optional)* If you want to have additional OpenVault clients that are not DMF servers, such as for running administrative commands, install the OpenVault software on those clients and run ov_admin as shown below. When asked for the server hostname, specify the virtual hostname. This connects the clients to the

virtual cluster, rather than a fixed host, so that upon migration they follow the server.

Note: You may wish to set the environment variable `OVSERVER` to the virtual hostname so that you can use the OpenVault administrative commands without having to specify the `-S` parameter on each command.

Do the following for each OpenVault client:

a. On the initial node:

   To allow an alternate node to act as an administrative client, run `ov_admin` and select the following menus, answering the questions when prompted:

```
initial# ov_admin
...
23 - Manage Applications and Admin CLI Tools
        7 - Activate a Host to use the Admin CLI Tools
...
```

b. On the OpenVault client node:

   Use `ov_admin` to enable the node to issue administrative commands by selecting:

```
client# ov_admin
...
31 - Enable Administrative Commands
```

Testing the `openvault` Resource

To test the OpenVault resource as part of a resource group named `dmfGroup`, do the following:

1. Move the resource group containing the `openvault` resource from the initial node to an alternate node:

   initial# **crm_resource -t group -r dmfGroup -M -H *alternate_node***

2. Verify that all the tape drives become available after a few moments. For example:

```
alternate# ov_stat -ld
Library Name   Broken   Disabled   State     LCP State
L700A          false    false      ready     ready
SL500-2        false    false      ready     ready

Drive Name     Group      Access Broken Disabled SoftState HardState DCP State   Occupied  Cartridge PCL
9940B_25a1     9940B_drives true false  false    ready     unloaded  ready       false
9940B_93c8     9940B_drives true false  false    ready     unloaded  ready       false
9940B_b7ba     9940B_drives true false  false    ready     unloaded  ready       false
LTO2_682f      LTO2_drives true  false  false    ready     unloaded  ready       false
LTO2_6832      LTO2_drives true  false  false    ready     unloaded  ready       false
LTO2_6835      LTO2_drives true  false  false    ready     unloaded  ready       false
LTO2_6838      LTO2_drives true  false  false    ready     unloaded  ready       false
```

3. Move the resource group containing the `openvault` resource back to the initial node:

   initial# **crm_resource -t group -r dmfGroup -M -H *initial_node***

4. Verify that all the tape drives become available after a few moments. For example:

```
initial# ov_stat -ld
Library Name   Broken   Disabled   State     LCP State
L700A          false    false      ready     ready
SL500-2        false    false      ready     ready

Drive Name     Group      Access Broken Disabled SoftState HardState DCP State   Occupied  Cartridge PCL
9940B_25a1     9940B_drives true false  false    ready     unloaded  ready       false
9940B_93c8     9940B_drives true false  false    ready     unloaded  ready       false
9940B_b7ba     9940B_drives true false  false    ready     unloaded  ready       false
LTO2_682f      LTO2_drives true  false  false    ready     unloaded  ready       false
LTO2_6832      LTO2_drives true  false  false    ready     unloaded  ready       false
LTO2_6835      LTO2_drives true  false  false    ready     unloaded  ready       false
LTO2_6838      LTO2_drives true  false  false    ready     unloaded  ready       false
```

5. Remove the resource location constraints generated by the administrative move commands executed above:

```
initial# crm_resource -t group -r dmfGroup -U
```

Configuring TMF for HA

To configure TMF for HA and create the TMF resource, do the following:

1. Copy the following file from the initial node to all of the other nodes that will act as DMF server nodes:

   ```
   /etc/tmf/tmf.config
   ```

   On each alternate node, if the tape drive pathname for a given drive is not the same as the pathname for the same drive on the initial node, modify the pathname in the `/etc/tmf.config` file on the alternate node so that it points to the appropriate pathname.

2. On each alternate node, execute the following to enable TMF startup during the boot process:

   ```
   alternate# chkconfig -s tmf on
   ```

3. Write an XML file for the TMF resource. Example 7-8 shows a TMF resource instance in a file named `tmf_resource.xml`.

Example 7-8 TMF Resource XML

```
<group id="dmfGroup">
<primitive id="tmf_resource" class="ocf" provider="sgi" type="tmf">
    <instance_attributes>
        <attributes>
            <nvpair name="devgrpnames" value="ibm3592,stk9940b"/>
            <nvpair name="mindevsup" value="2,5"/>
            <nvpair name="devtimeout" value="109,118"/>
            <nvpair name="admin_emails" value="admin1,admin2"/>
            <nvpair name="loader_names" value="ibm3494,l700a"/>
            <nvpair name="loader_hosts" value="ibm3494cps,stk9710"/>
            <nvpair name="loader_users" value="root,acsss"/>
            <nvpair name="loader_passwords" value="my!pass,myOtherPass"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="236s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="236s" on_fail="fence"/>
        <op name="monitor" interval="60s" timeout="30s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
</group>
```

The tag elements and their attributes are as follows:

- `group id` specifies the name of the resource group to which this resource will belong, in this case `dmfGroup`.
- `primitive id` specifies the name of the resource, in this case `tmf_resource`. The element's attributes must be of class `ocf`, provider `sgi`, and type `tmf`.
- `devgrpnames` is a comma-separated list of TMF device groups defined in the `tmf.config` file that are to be managed by Heartbeat. In this case, `ibm3592` and `stk9940b`.

- `mindevsup` is a comma-separated list of the number of devices, one per device group, that must be configured up successfully within the corresponding device group in order to count the group as being highly available. In this case, at least two devices in the `ibm3592` device group and five devices in the `stk9940b` device group must be up. (A value of `0` indicates that failover will never be initiated, even if all the devices in that device group are unavailable.)

- `devtimeout` is a comma-separated list of timeouts in seconds, one per device group, that are used to decide how long to wait for a device in that device group to finish configuring up or down. In this case, the timeouts are 109 and 118 seconds, respectively.

  Changing the up/down state of a device may require rewinding and unloading a tape left in the drive by a previous host. Different tape device types have different maximum rewind and unload times, which can be obtained from the vendor's product literature. The timeout value for a particular device group should be calculated by adding the maximum rewind time for a device in that group to the device's unload time, then adding an additional 10 seconds to allow for any required robot hand movement.

  The XML example shows 3592 tape drives with a maximum rewind time of 78 seconds and an unload time of 21 seconds, resulting in a `devtimeout` value of 78+21+10=109 seconds. It also shows 9940B tape drives with a maximum rewind time of 90 seconds and an unload time of 18 seconds, resulting in a `devtimeout` of 90+18+10=118.

- `admin_emails` is a comma-separated list of administrator email addresses corresponding to the device groups listed in `devgrpnames`, in this case `admin1` and `admin2`. The same email address can be used for more than one device group (such as "`admin1,admin1`"). The email address will be used to send a message whenever tape drives that were previously available become unavailable, so that the administrator can take action to repair the drives in a timely fashion.

- `loader_names` is a comma-separated list of loader names configured in `tmf.config` that correspond to the device groups listed in `devgrpnames`, in this case `ibm3494` and `l700a`.

- `loader_hosts` is a comma-separated list of hosts through which the corresponding loaders listed in `loader_names` are controlled, in this case `ibm3494cps` and `stk9710`.

- `loader_users` is a comma-separated list of user names that are used to log in to the corresponding hosts listed in `loader_hosts`, in this case `root` and `acsss`.
- `loader_passwords` a comma-separated list of passwords corresponding to the user names listed in `loader_users`, in this case `my!pass` and `myOtherPass`.
- `start` is the name of the operation that initiates the resource. It requires that fencing is configured and active in order to start the resource. Using system reset as a fencing method is required in order to preserve data integrity. If the `start` operation fails, it will attempt to restart the resource.

  The `tmf` resource agent will try twice to configure each drive up before considering it unusable, so the `start timeout` value should therefore be at least twice the greatest `devtimeout` value. In the XML example, the start timeout should be 2*118=236.
- `stop` is the name of the operation that terminates or gives up control of the resource. You should use the same `timeout` value for both the `stop` and `start` operations. Therefore, in this case, it will timeout after 236 seconds. If the stop operation fails, it will attempt to fence the node on which the failure occurred. The `stop` fail policy must be set to `fence` and a STONITH facility must be configured according to the requirements for your site.
- `monitor` is the name of the operation that determines if the resource is operating correctly. The resource will be monitored at an interval of 60 seconds. Each `monitor` operation will timeout after 30 seconds. If the monitor operation fails, it will attempt to restart the resource.
- `resource_stickiness` specifies a weight for the preference to keep this resource on the node on which it is currently running. A positive value specifies a preference for the resource to remain on the node on which it is currently running. This preference may only be overridden if the node becomes ineligible to run the resource (if the node goes into standby mode) or if there is a `start`, `monitor`, or `stop` failure for this resource or another resource in the same resource group.
- `resource_failure_stickiness` specifies a weight for the preference to move this resource to a new node based on the number of `start`, `monitor`, or `stop` failures that this resource has experienced. The value of `-INFINITY` specifies that if this resource experiences a failure, this resource and all members of its resource group will be restarted on a different eligible node.

4. Include the resource in the CIB. For example:

```
initial# cibadmin -U -o resources -x tmf_resource.xml
```

Testing the tmf Resource

To test the TMF resource as part of a resource group named dmfGroup, do the following:

1. Use tmstat to verify that all of the tape drives in all Heartbeat-managed device groups are in assn or idle status on the initial node.
2. Move the resource group containing the tmf resource to an alternate node:

```
initial# crm_resource -t group -r dmfGroup -M -H alternate_node
```

3. Verify that the state is correct:
   - Use tmstat to verify that the tape drives all have a status of down or sdwn on the initial node and that they have a status of idle or assn on the alternate node
   - Use tmmls to verify that all of the loaders on the initial node still have a status of UP
4. Verify that the timeout values for the start, stop, and monitor operations are appropriate. Do the following:
   a. On the alternate node, look in /var/log/messages for the time when the resource start operation started and ended. Also capture the start and end times of the monitor operation.
   b. On the initial node, look in /var/log/messages to find the start and stop times for the stop operation.
   c. Subtract the ending time from the starting time in each case to get the required time for each operation.
   d. Take the above values and increase them by 10%.

Following are examples of finding the `start`, `stop`, and `monitor` operation durations:

```
initial# egrep "Performing op=tmf_resource_start|operation tmf_resource_start.*complete" /var/log/messages
Jul 22 16:30:26 node-1 crmd: [4786]: info: do_lrm_rsc_op: Performing op=tmf_resource_start_0
key=34:31:04c19af8-e90d-43e6-b2fb-3b05d097f338)
Jul 22 16:30:30 node-1 crmd: [4786]: info: process_lrm_event: LRM operation tmf_resource_start_0 (call=367, rc=0)
complete

initial# egrep "Performing op=tmf_resource_stop|operation tmf_resource_stop.*complete" /var/log/messages
Jul 20 20:05:34 node-1 crmd: [4786]: info: do_lrm_rsc_op: Performing op=tmf_resource_stop_0
key=9:4:04c19af8-e90d-43e6-b2fb-3b05d097f338)
Jul 20 20:05:34 node-1 crmd: [4786]: info: process_lrm_event: LRM operation tmf_resource_stop_0 (call=34, rc=0)
complete

initial# egrep "Performing op=tmf_resource_monitor|operation tmf_resource_monitor.*complete" /var/log/messages
Jul 22 16:30:22 node-1 crmd: [4786]: info: do_lrm_rsc_op: Performing op=tmf_resource_monitor_0
key=13:31:04c19af8-e90d-43e6-b2fb-3b05d097f338)
Jul 22 16:30:22 node-1 crmd: [4786]: info: process_lrm_event: LRM operation tmf_resource_monitor_0 (call=349, rc=7)
complete
```

5. If you need to change the timeout values, edit the original XML file and update the database with the modified XML. For example, for the local XVM resource:

   ```
   initial# cibadmin -U -o resources -x local_xvm.xml
   ```

6. Move the resource group containing the `tmf` resource back to the initial node:

   ```
   initial# crm_resource -t group -r dmfGroup -M -H initial_node
   ```

7. Verify that the state is correct:

   - Use `tmstat` to verify that the tape drives all have a status of `down` or `sdwn` on the alternate node and that they have a status of `idle` or `assn` on the initial node
   - Use `tmmls` to verify that all of the loaders on the alternate node still have a status of `UP`

8. Remove the resource location constraints generated by the administrative move commands executed above:

   ```
   initial# crm_resource -t group -r dmfGroup -U
   ```

Configuring DMF for HA

To configure DMF for HA, do the following:

1. Make the filesystem backup inventory accessible from all DMF servers in the Heartbeat cluster.

   Backup of DMF-managed user filesystems and DMF administrative filesystems is always performed on the active DMF server based upon parameters in the DMF configuration file. The `xfsdump` command maintains an inventory of all backups performed within the directory `/var/lib/xfsdump`, but in an HA environment, the active DMF server node can change over time. Therefore, in order for `xfsdump` to maintain a consistent inventory, it must be able to access the inventory for all past backups even if those backups were created on another node.

   SGI recommends that you make the inventory accessible to all DMF server nodes by relocating it into a Heartbeat-managed DMF administrative filesystem within the same resource group as DMF. For example, create a site-specific directory in DMF's `HOME_DIR`, such as `/dmf/home/site_specific`:

   - On the initial node currently containing the inventory, enter the following:

     ```
     initial# cd /var/lib
     initial# cp -r xfsdump /dmf/home/site_specific/xfsdump
     initial# mv xfsdump xfsdump.bak
     initial# ln -s /dmf/home/site_specific/xfsdump xfsdump
     ```

     For example:

   - On all alternate DMF server nodes in the cluster, enter the following:

     ```
     alternate# cd /var/lib
     alternate# mv xfsdump xfsdump.bak
     alternate# ln -s /dmf/home/site_specific/xfsdump xfsdump
     ```

Note: It is the `/var/lib/xfsdump` directory that should be shared, rather than the `/var/lib/xfsdump/inventory` directory. If there are inventories stored on various nodes, you can use `xfsinvutil` to merge them into a single common inventory, prior to sharing the inventory among the nodes in the cluster.

---

2. On the initial node, modify the DMF `dmf.conf` configuration file as follows:

   - Set the `MAX_MS_RESTARTS` parameter in the appropriate drive group stanzas to `0` so that DMF will not restart the mounting service.
   - Set the `DUMP_INVENTORY_COPY` parameter to use only a DMF HA administrative filesystem on a different disk than the live inventory created above. If the live inventory in `/dmf/home/site_specific/xfsdump` is lost, you can then recreate it from the inventory backup in `DUMP_INVENTORY_COPY`. For example, you could create the directory `/dmf/journal/site_specific/inventory_copy` for use in `DUMP_INVENTORY_COPY`.
   - If you are using OpenVault, set the `MSG_DELAY` parameter in the `drivegroup` stanzas to a value of slightly more than 2 minutes.

   For more information, see the `dmf.conf`(5) man page and the *DMF Administrator's Guide for SGI InfiniteStorage*.

3. Copy the DMF configuration file (`/etc/dmf/dmf.conf`) from the initial node to all of the other DMF servers in the Heartbeat cluster.

4. Disable the automatic start of DMF during boot on all DMF server nodes in the Heartbeat cluster:

   ```
   # chkconfig -s dmf off
   ```

5. Write the XML for the DMF resource. Example 7-9 shows a DMF resource instance in a file named `dmf_resource.xml`.

Example 7-9 DMF Resource XML

```
<group id="dmfGroup">
<primitive id="dmf_resource" class="ocf" provider="sgi" type="dmf">
    <instance_attributes>
        <attributes>
            <nvpair name="admin_email_addresses" value="root,admin1"/>
            <nvpair name="only_email_on_failure" value="false"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="30s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="30s" on_fail="fence"/>
        <op name="monitor" interval="60s" timeout="30s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
</group>
```

The tag elements and their attributes are as follows:

- `group id` specifies the name of the resource group to which this resource will belong, in this case `dmfGroup`.
- `primitive id` specifies the name of the resource, in this case `dmf_resource`. The element's attributes must be of class `ocf`, provider `sgi`, and type `dmf`.
- `admin_email_addresses` is the name of an attribute that takes a comma-separated list of email addresses to be notified in the case of errors. In this case, the users `root` and `admin1` will be notified.
- `only_email_on_failure` is the name of the attribute that determines whether or not to report DMF failures to Heartbeat. It can be set to `true` (meaning that a DMF failure **will not** result in a Heartbeat resource failure) or to `false`.

- `start` is the name of the operation that initiates the resource. It will timeout after 30 seconds. It requires that fencing is configured and active in order to start the resource. Using system reset as a fencing method is required in order to preserve data integrity. If the `start` operation fails, it will attempt to restart the resource.
- `stop` is the name of the operation that terminates or gives up control of the resource. It will timeout after 30 seconds. If the stop operation fails, it will attempt to fence the node on which the failure occurred. The `stop` fail policy must be set to `fence` and a STONITH facility must be configured according to the requirements for your site.
- `monitor` is the name of the operation that determines if the resource is operating correctly. The resource will be monitored at an interval of 60 seconds. Each `monitor` operation will timeout after 30 seconds. If the monitor operation fails, it will attempt to restart the resource.
- `resource_stickiness` specifies a weight for the preference to keep this resource on the node on which it is currently running. A positive value specifies a preference for the resource to remain on the node on which it is currently running. This preference may only be overridden if the node becomes ineligible to run the resource (if the node goes into standby mode) or if there is a `start`, `monitor`, or `stop` failure for this resource or another resource in the same resource group.
- `resource_failure_stickiness` specifies a weight for the preference to move this resource to a new node based on the number of `start`, `monitor`, or `stop` failures that this resource has experienced. The value of `-INFINITY` specifies that if this resource experiences a failure, this resource and all members of its resource group will be restarted on a different eligible node.

6. Include the resource in the CIB. For example:

```
initial# cibadmin -U -o resources -x dmf_resource.xml
```

Testing the dmf Resource

To test the dmf resource as part of a resource group named dmfGroup, do the following:

1. Verify that DMF has started by using the dmdstat -v command and manual dmput and dmget commands on the initial node:

   ```
   initial# dmdstat -v
   initial# dmput test_file
   initial# dmdidle
   (wait a bit to allow time for the tape to be written and unmounted)
   initial# dmget test_file
   ```

2. Move the resource group containing the dmf resource to an alternate node (because the mounting service is in the same resource group, it must be co-located and thus should failover with DMF to the new node):

   ```
   initial# crm_resource -t group -r dmfGroup -M -H alternate_node
   ```

3. Verify that DMF has started on the new node by using the dmdstat -v command and manual dmput and dmget commands on the alternate node:

   ```
   alternate# dmdstat -v
   alternate# dmput test_file
   alternate# dmdidle
   (wait a bit to allow time for the tape to be written and unmounted)
   alternate# dmget test_file
   ```

4. Move the resource group containing the dmf resource back to the initial node:

   ```
   initial# crm_resource -t group -r dmfGroup -M -H initial_node
   ```

5. Verify that DMF has started by using the dmdstat -v command and manual dmput and dmget commands on the initial node:

   ```
   initial# dmdstat -v
   initial# dmput test_file
   initial# dmdidle
   (wait a bit to allow time for the tape to be written and unmounted)
   initial# dmget test_file
   ```

6. Remove the resource location constraints generated by the administrative move commands executed above:

```
initial# crm_resource -t group -r dmfGroup -U
```

Configuring NFS for HA

To configure NFS for HA, do the following:

1. Copy the `/etc/exports` entries that you would like to make highly available from the initial node to the `/etc/exports` file on the alternate node.
2. Disable the NFS server from starting automatically on boot on both the initial node and the alternate node:

```
initial# chkconfig nfsserver off
alternate# chkconfig nfsserver off
```

3. Write the XML for the NFS resource. Example 7-10 shows a DMF resource instance in a file named `dmf_resource.xml`.

Example 7-10 NFS Resource XML

```
<group id="dmfGroup">
<primitive id="nfs_resource" class="lsb" type="nfsserver">
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
</group>
```

The tag elements and their attributes are as follows:

- `group id` specifies the name of the resource group to which this resource will belong, in this case `dmfGroup`.
- `primitive id` specifies the name of the resource, in this case `nfs_resource`. The element's attributes must be of class `lsb` and type `nfsserver`.
- `resource_stickiness` specifies a weight for the preference to keep this resource on the node on which it is currently running. A positive value

specifies a preference for the resource to remain on the node on which it is currently running. This preference may only be overridden if the node becomes ineligible to run the resource (if the node goes into standby mode) or if there is a `start`, `monitor`, or `stop` failure for this resource or another resource in the same resource group.

- `resource_failure_stickiness` specifies a weight for the preference to move this resource to a new node based on the number of `start`, `monitor`, or `stop` failures that this resource has experienced. The value of `-INFINITY` specifies that if this resource experiences a failure, this resource and all members of its resource group will be restarted on a different eligible node.

4. Include the resource in the CIB. For example:

```
initial# cibadmin -U -o resources -x nfs_resource.xml
```

For additional information, see:

http://linux-ha.org/LSB
http://linux-ha.org/LSBResourceAgent

Testing the `nfsserver` Resource

To test the `nfsserver` resource in a resource group named `dmfGroup`, do the following:

1. Run the following command on the initial node to verify that the NFS filesystems are exported:

```
initial# exportfs -v
/work.mynode1   <world>(rw,wdelay,root_squash,no_subtree_check)
/work.mynode2   <world>(rw,wdelay,root_squash,no_subtree_check)
/work.mynode3   <world>(rw,wdelay,root_squash,no_subtree_check)
/work.mynode4   <world>(rw,wdelay,root_squash,no_subtree_check)
/mirrors        <world>(ro,wdelay,root_squash,no_subtree_check)
/               <world>(ro,wdelay,root_squash,no_subtree_check)
```

2. Mount the filesystems on a node that will not be a DMF server (`third`):

```
third# mount initial:/nfsexportedfilesystem /mnt/test
```

3. Read and write to the NFS-mounted filesystems:

```
third# echo "test data for a test file" > /mnt/test/testFile1A
third# cat /mnt/test/testFile1A
test data for a test file
```

4. Move the resource group containing the nfsserver resource from the initial node to an alternate node:

```
initial# crm_resource -t group -r dmfGroup -M -H alternate_node
```

5. Run the following command on the alternate node to verify that the NFS filesystems are exported:

```
alternate# exportfs -v
/work.mynode1   <world>(rw,wdelay,root_squash,no_subtree_check)
/work.mynode2   <world>(rw,wdelay,root_squash,no_subtree_check)
/work.mynode3   <world>(rw,wdelay,root_squash,no_subtree_check)
/work.mynode4   <world>(rw,wdelay,root_squash,no_subtree_check)
/mirrors        <world>(ro,wdelay,root_squash,no_subtree_check)
/               <world>(ro,wdelay,root_squash,no_subtree_check)
```

6. Read and write to the NFS-mounted filesystems:

```
third# echo "test data for another test file" > /mnt/test/testFile1B
third# cat /mnt/test/testFile1B
test data for another test file
```

7. Move the resource group containing the nfsserver resource back to the initial node:

```
initial# crm_resource -t group -r dmfGroup -M -H initial_node
```

8. Remove the resource location constraints generated by the administrative move commands executed above:

```
initial# crm_resource -t group -r dmfGroup -U
```

Configuring and Testing STONITH Reset Services

Using system reset as a fencing method is required in order to preserve data integrity:

- For Altix 450 systems and Altix 4700 systems using L2 controllers, SGI provides the `l2network` reset service agent
- For Altix XE x86_64 systems with a baseboard management controller (BMC) using intelligent platform management interface (IPMI) network reset, SGI provides the `sgi-ipmi` reset service agent

---

Note: The STONITH reset service on Altix XE x85_64 requires the use of a BMC user account with administrative privileges. For more information, see the `ipmitool`(1) man page, *Guide to Programming Environments and Tools Available on SGI Altix XE System*, and the user guide or quick start guide for your system.

This chapter discusses creating the STONITH reset services using the XML method, testing them, and enabling reset services:

Configuring L2 STONITH Reset for Altix ia64 Systems

To configure L2 STONITH reset, do the following:

1. Write an XML file for the L2 STONITH reset service. Example 8-1 shows an L2 STONITH reset service for an Altix 450 or Altix 4700 system in a file named `stonith-l2network-set.xml`.

Example 8-1 `l2network` Altix 450 or Altix 4700 System STONITH Reset Service XML

```
<clone id="stonith-l2network-set">
  <instance_attributes>
    <attributes>
      <nvpair name="clone_max" value="2"/>
      <nvpair name="clone_node_max" value="1"/>
    </attributes>
  </instance_attributes>
  <primitive id="stonith-l2network" class="stonith" type="l2network">
    <instance_attributes>
      <attributes>
        <nvpair name="nodelist" value="node-0;128.162.245.170;;3 node-1;128.162.245.170;;4"/>
      </attributes>
    </instance_attributes>
    <operations>
      <op name="start" timeout="5s" prereq="nothing" on_fail="restart"/>
      <op name="monitor" timeout="5s" interval="10s" prereq="nothing" on_fail="restart"/>
    </operations>
  </primitive>
</clone>
```

The tag elements and their attributes are as follows:

- `clone id` is the name of the clone set, in this case `stonith-l2network-set`.
- `clone_max` is the maximum number of copies of this reset service that may be run in the cluster (must be `2` for a two–node cluster).
- `clone_node_max` is the maximum number of copies that may run on any given node (must be `1`).
- `primitive id` specifies the name of the reset service that is cloned, in this case `stonith-l2network`. The element's attributes must be of class `stonith` and type `l2network`.

- `nodelist` specifies the nodes to be acted upon, providing the following information for each node (each field is separated by a semicolon):

  nodename ; L2_ipaddr ; L2passwd ; partition

  where the fields are:

  – Node name (such as `node-0`)

  – L2 IP address (such as `128.162.245.170`)

  – L2 password (the empty field in the XML example above indicates that the L2 has no password)

  – Machine partition (such as `3`) ; use `0` for a nonpartitioned system, which is the most common circumstance

  ---

  Note: The following command shows the partition ID on an Altix ia64 system:

  ```
  # cat /proc/sgi_sn/partition_id
  ```

- `start` is the name of the operation that initiates or gains control of the reset service. It will timeout after 5 seconds. If the `start` operation fails, it will attempt to restart the reset service.

- `monitor` is the name of the operation that determines if the reset service is operating correctly. The reset service will be monitored at an interval of 10 seconds. Each `monitor` operation will timeout after 5 seconds. If the `monitor` operation fails, it will attempt to restart the reset service.

2. Include the reset service in the CIB. For example:

```
initial# cibadmin -C -o resources -x stonith-l2network-set.xml
```

For more information, see the *SGI L1 and L2 Controller Software User's Guide* and the user guide or quick start for your system.

Testing the `l2network` Reset Service

To test the L2 STONITH reset service, do the following;

1. Enter the following command:

```
initial# stonith -t l2network -T reset nodelist="nodelist" machine_to_reset
```

For example, to reset `node-0` (line breaks shown for readability):

```
initial# stonith -t l2network -T reset \
nodelist="node-0;128.162.245.170;;3 node-1;128.162.245.170;;4" node-0
** INFO: Initiating l2network-reset on node0 via L2 128.162.245.170, partition 3
```

2. Verify that the specified node was reset and was able to successfully complete a reboot.

Configuring IPMI STONITH Reset Service for Altix XE x86_64 Systems

To configure IPMI reset, do the following:

1. Write an XML file for the IPMI STONITH reset service. Example 8-2 shows an SGI Altix XE 310 system.

Example 8-2 `sgi-ipmi`Altix XE 310 System Reset Service XML

```
<clone id="stonith-sgi-ipmi-set">
  <instance_attributes>
    <attributes>
      <nvpair name="clone_max" value="2"/>
      <nvpair name="clone_node_max" value="1"/>
    </attributes>
  </instance_attributes>
  <primitive id="stonith-sgi-ipmi" class="stonith" type="external/sgi-ipmi">
    <instance_attributes>
      <attributes>
        <nvpair name="nodelist" value="node-0;admin;admin;supermicro;128.162.245.170
                node-1;admin;admin;supermicro;128.162.245.171"/>
      </attributes>
    </instance_attributes>
    <operations>
      <op name="start" timeout="5s" prereq="nothing" on_fail="restart"/>
      <op name="monitor" timeout="5s" interval="10s" prereq="nothing" on_fail="restart"/>
```

```
    </operations>
  </primitive>
</clone>
```

The tag elements and their attributes are as follows:

- `clone id` is the name of the clone set, in this case `stonith-sgi-ipmi-set`.
- `clone_max` is the maximum number of copies of this reset service that may be run in the cluster (must be `2` for a two–node cluster).
- `clone_node_max` is the maximum number of copies that may run on any given node (must be `1`).
- `primitive id` specifies the name of the reset service that is cloned, in this case `stonith-sgi-ipmi`. The element's attributes must be of class `stonith` and type `external/sgi-ipmi`.
- `nodelist` specifies the nodes to be acted upon, providing the following information for each node (each field is separated by a semicolon):

  nodename ; *userID* ; *IPMIpasswd* ; *BMCtype* ; *IPMIipaddr1* [; *IPMIipaddrN*]

  where the fields are:

  – Node name (such as `node-0`)

  – User ID to use on the IPMI device (such as the default `admin`)

  – IPMI device password (such as the default `admin`)

  – BMC type of the IPMI device:

    - `intel` for Altix XE240 using Intel
    - `supermicro` for Altix XE310, Altix XE250, or Altix XE320 using Supermicro

  – IPMI device IP address (such as `128.162.245.170` and `128.162.245.171`)
- `start` is the name of the operation that initiates or gains control of the reset service. It will timeout after 5 seconds. If the `start` operation fails, it will attempt to restart the reset service.
- `monitor` is the name of the operation that determines if the reset service is operating correctly. The reset service will be monitored at an interval of 10

seconds. Each `monitor` operation will timeout after 5 seconds. If the `monitor` operation fails, it will attempt to restart the reset service.

2. Include the reset service in the CIB. For example:

```
initial# cibadmin -C -o resources -x stonith-sgi-ipmi-set.xml
```

Testing the `external/sgi-ipmi` Reset Service

To test the IPMI STONITH reset service, do the following:

1. Enter the following command:

```
initial# stonith -t external/sgi-ipmi -p "nodelist_value" -T reset node_to_be_reset
```

For example, to reset `node1`:

```
initial# stonith -t external/sgi-ipmi -p "node-0;admin;admin;supermicro;128.162.245.170" -T reset node-0
```

2. Verify that the specified node was reset and was able to successfully complete a reboot.

Enabling System Reset

To enable system reset (which is disabled by default), enter the following:

```
initial# crm_attribute -t crm_config -n stonith-enabled -v true
```

Note: After all of the reset services are configured and tested, ensure that any constraints remaining in the cluster are appropriate for a production environment. To remove any remaining hostname constraint, enter the following:

```
initial# crm_resource -t group -r dmfGroup -r -U
```

HA Administrative Tasks and Considerations

This chapter discusses various administrative tasks and considerations for an HA cluster:

Understanding CIFS and NFS in a Heartbeat Cluster

CIFS failover requires that the client application reissue the I/O after the failover occurs. Applications such as XCOPY will do this, but many other applications will not. Applications that do not retry may abort when CIFS services are moved between nodes.

NFS failover is handled by the kernel, so no changes are required for an NFS client application; applications doing I/O on NFS will pause while the failover is occurring.

Using the DMF `run_daily_drive_report` Task

The `run_daily_drive_report` DMF task tells you about drives that need cleaning. It uses `tsreport` and `ts` log files that are on the local host. Therefore, it will only

tell you about cleaning notifications that are in files on the local host. It does not make any attempt to copy the `ts` log files from the alternate node.

For example, suppose `machineA` was the DMF server from 00:01 (12:01 am) until 12:00 (noon), when a failover to `machineB` occurred. Sometime during the morning, `driveB` reported that it needed cleaning. If `run_daily_drive_report` ran at 12:30 (12:30 pm) and `driveB` had not been used in that half hour, `run_daily_drive_report` would not report that `driveB` needed cleaning.

To work around this problem, use the `tsreport` command on both nodes.

Reviewing Log Files

You will find information about Heartbeat in the following log files:

Log File	Contents
`/var/log/ha_log`	Heartbeat daemon warning and error messages
`/var/log/ha_debug`	All Heartbeat daemon messages
`/var/log/messages`	All other executable messages, such as that from resource agents

You can set the debugging level in the `/etc/ha.d/ha.cf` file.

Caution: Values `2` and larger can have a negative effect on the logfile size and on cluster reliability.

Making a Backup Copy of the CIB

When you have completed your configuration, make a backup copy of the cluster information base (CIB) so that you can return to it if necessary. For example:

```
# cibadmin -Q > CIB_backup_filename
```

Restoring a Previous Configuration

If you have previously made a backup copy of the cluster information base (CIB), you can restore it as the current CIB by doing the following:

1. Erase the existing corrupted CIB:

   # **cibadmin -E**

2. Create a new CIB from the backup copy:

   # **cibadmin -C -x *CIB_backup_file***

Editing Resources

Following is one method to edit existing resources:

1. Extract the current resources and put them into a file named `resources.orig`:

   # **cibadmin --obj_type resources --cib_query > resources.orig**

2. Make a copy of the resources that you can edit (allowing you to keep the `resources.orig` available as a backup if needed):

   # **cp resources.orig resources.new**

3. Edit the XML values that you wish to change in `resources.new`, such as by using the `vi` editor:

   # **vi resources.new**

4. Feed the changed contents of `resources.new` back into the CIB:

```
# cibadmin --obj_type resources --cib_update --xml-pipe < resources.new
```

5. Extract the database information again and compare the original to the new database state:

```
# cibadmin --obj_type resources --cib_query > resources.modified
# diff -u resources.orig resources.modified
```

Clearing the Fail Count for a Resource

To clear the fail-count for a resource, enter the following:

crm_failcount -U *node_where_failure_occurred* -r *resource_that_failed* -D

Clearing the fail count contributes to making the resource and members of its resource group eligible to move back to the original preferred node. However, the ability to move depends upon other factors, such as other failures, resource location constraints, and resource stickiness values.

Manually Issuing a System Reset

To manually issue a system reset, do the following:

- For SGI Altix ia64 Systems, where *nodelist_value* is the value for the `nodelist` attribute as described in "Configuring L2 STONITH Reset for Altix ia64 Systems" on page 72:

  `stonith -t l2network -p "`*nodelist_value*`" -T reset` *node_to_be_reset*

  For example, to reset `node-0`:

  ```
  # stonith -t l2network -p "node-0;128.162.245.170;;3" -T reset node-0
  ```

  In the above command, `128.162.245.170` is the IP address of the L2 that has `node-0` configured as partition 3.

- For IPMI reset on Altix XE x86_64 systems, where *nodelist_value* is the value of the `nodelist` attribute in "Configuring IPMI STONITH Reset Service for Altix XE x86_64 Systems" on page 74:

  `stonith -t external/sgi-ipmi -p "`*nodelist_value*`" -T reset` *node_to_be_reset*

  For example, to reset `node-0`:

```
# stonith -t external/sgi-ipmi -p "node-0;admin;admin;supermicro;128.162.245.170" -T reset node-0
```

  In the above command, `node-0` has a BMC responding at IP address `128.162.245.170`. The BMC is a Supermicro and has been configured with usercode `admin` and password `admin`.

  If you enter the above command on `node-0`, it will reboot `node-0`. If you execute the command from another node, it will execute the IPMI power-off and power-on commands via the BMC at `128.162.245.170`.

In general, the `external/sgi-ipmi` STONITH agent will execute the reboot command if it is run on the node that will be reset or it will execute the IPMI

power-off and power-on commands via the first responsive BMC at one of the IP addresses provided in *nodelist_value.*

Removing Heartbeat Control of a Resource Group

To remove Heartbeat control for a resource group, set the `is_managed` parameter to `false`, enter the following:

```
# crm_resource -r resourcegroup_name -p is_managed -v false
```

To return control of the resource to Heartbeat, enter the following:

```
# crm_resource -r resourcegroup_name -p is_managed -v true
```

Stopping Heartbeat

To stop Heartbeat on the local node, enter the following:

```
# service heartbeat stop
```

Troubleshooting

This section discusses the following:

For details about troubleshooting Heartbeat, see the documentation referred to in "Sources for Detailed Heartbeat Documentation" on page 5.

General Heartbeat Troubleshooting

If you notice problems with Heartbeat, do the following:

1. Watch the output from `crm_mon`.
2. If things do not seem to be responding correctly or if `crm_mon` lists an error, execute the following:

   ```
   # crm_verify -LV
   ```

   Note: You can run `crm_verify` with a larger number of `-V` command-line arguments for more detail. If you run `crm_verify` before STONITH is enabled, you will see errors. Errors similar to the following may be ignored at this time and will go away after STONITH is configured (line breaks used here for readability):

   ```
   crm_verify[182641]: 2008/07/11_16:26:54 ERROR: unpack_operation:
   Specifying on_fail=fence and
   stonith-enabled=false makes no sense
   ```

3. If there are any problems listed in the `crm_verify` output, they will contain the failed action and the host on which the action failed. To find the specific problem,

try to match those events to messages in /var/log/messages or to other information you have about the cluster state.

Recovering from an Incomplete Failover

After an incomplete failover, in which one or more of the resources are not started and the cluster can no longer provide high availability, you must do the following to restore resource functionality and high availability:

1. Disable Heartbeat management from the resource group:

```
# crm_resource -r  resourcegroupname  -p is_managed -v false
```

2. Determine which resources have failcounts:

   ```
   # crm_failcount -G -U nodename  -r resourcename
   ```

   Repeat for each resource on each node.

3. Troubleshoot the failed resource operations. Examine the /var/log/messages system log and application logs around the time of the operation failures in order to deduce why they failed and deal with those causes.

4. Ensure that all of the individual resources are working properly according to the information in Chapter 7, "Configuring SGI Products for High Availability and Testing" on page 31.

5. Remove the failcounts found in step 2:

   ```
   # crm_failcount -D -U nodename  -r failed_resourcename
   ```

   Repeat this for each failed resource on each node.

6. Remove error messages:

   ```
   # crm_resource -C -H nodename -r failed_resourcename
   ```

   Repeat this for each failed resource on each node.

7. Reenable Heartbeat management:

   ```
   # crm_resource -r resourcegroupname -d is_managed
   ```

Error Messages in `/var/log/messages`

If you see errors in the `/var/log/messages` file, you should run the `crm_verify` command and look for corresponding messages.

`dmaudit` Error Detection

Files in the DMF-managed user filesystems that are out of synchronization with DMF database entries because of a system crash may cause errors to be found by `dmaudit`.

For example, when a user file is removed, the DMF daemon will immediately soft-delete the database entries corresponding to that file indicating at what time the file was removed. Should the machine crash shortly thereafter, the removal of the file within the filesystem might not have yet been updated on disk, and so the file will reappear when the filesystem is mounted on the alternate node. If `dmaudit` is then run, it will report that the filesystem and database are inconsistent because it found an existing file pointing at a soft-deleted database entry. You can minimize these errors by using the `dirsync` mount option if potentially lower filesystem performance is not a concern.

DMF-managed user filesystems may be mounted with either the `sync` or `dirsync` mount option, depending on the your desire for filesystem completeness upon system failure. Alternatively, the user applications accessing these filesystems are responsible for verifying file synchronization.

Caution: The `sync` and `dirsync` mount options have serious performance implications that may outweigh filesystem synchronization benefits.

DMF Logs are Incomplete

DMF logs may not be complete due to system failures. If you want to ensure that the DMF logs are absolutely complete upon system failure, despite any performance issues, you may optionally choose to mount the DMF spool filesystem with the `sync` mount option.

The DMF home, journals, and disk MSP filesystems do not require the `sync` mount option because DMF synchronizes data on those filesystems. The move and temporary filesystems do not require the `sync` mount option because DMF does not recover anything from them.

Caution: The `sync` mount option has serious performance implications that may outweigh filesystem synchronization benefits.

Reporting Problems to SGI

If you need to report problems to SGI Support, do the following:

- Run the following command as `root` on every node in the cluster in order to gather system configuration information:

  # **/usr/sbin/system_info_gather -A -o *nodename*.out**

- Gather the information reported about Heartbeat in the logfiles; see "Reviewing Log Files" on page 78.

- Run the following command once on the node where the resource group currently resides to collect information for today and the specified number of additional days (*extra-days* must be a numerical value greater than or equal to `0`):

  # **dmcollect *extra-days***

  See the `dmcollect`(8) man page for additional information.

- *(If TMF is used)* Run the following command once on the node where the resource group currently resides:

  # **tmcollect [*directory*]**

  See the `tmcollect`(8) man page for additional information.

When you contact SGI Support, you will be provided with information on how and where to upload the collected information files for SGI analysis.

Complete XML Examples

This appendix contains the following:

Complete XML Example for DMF and TMF

Example A-1 shows a fully defined DMF and TMF service, not including cluster-wide parameters.

Example A-1 Complete XML Example for DMF and TMF

```
<group id="dmfGroup">
<instance_attributes id="dmfGroup">
    <attributes>
        <nvpair id="dmfGroup-collocated" name="collocated" value="true"/>
        <nvpair name="ordered" id="dmfGroup-ordered" value="true"/>
    </attributes>
</instance_attributes>
<primitive id="local_xvm" class="ocf" provider="sgi" type="lxvm">
    <instance_attributes>
        <attributes>
            <nvpair name="volnames" value="openvault,home,journals,spool,move,tmp,diskmsp,dmfusr1,dmfusr3"/>
            <nvpair name="physvols" value="myCluster,myClusterStripe1,myClusterStripe2"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="60s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="60s" on_fail="fence"/>
        <op name="monitor" interval="20s" timeout="40s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
```

```
    </meta_attributes>
</primitive>
<primitive id="openvaultFS" class="ocf" provider="heartbeat" type="Filesystem">
    <instance_attributes>
        <attributes>
            <nvpair name="device" value="/dev/lxvm/openvault"/>
            <nvpair name="directory" value="/dmf/openvault"/>
            <nvpair name="fstype" value="xfs"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="15s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="10s" on_fail="fence"/>
        <op name="monitor" interval="10s" timeout="10s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
<primitive id="homeFS" class="ocf" provider="heartbeat" type="Filesystem">
    <instance_attributes>
        <attributes>
            <nvpair name="device" value="/dev/lxvm/home"/>
            <nvpair name="directory" value="/dmf/home"/>
            <nvpair name="fstype" value="xfs"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="15s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="10s" on_fail="fence"/>
        <op name="monitor" interval="10s" timeout="10s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
```

```
</primitive>
<primitive id="spoolFS" class="ocf" provider="heartbeat" type="Filesystem">
    <instance_attributes>
        <attributes>
            <nvpair name="device" value="/dev/lxvm/spool"/>
            <nvpair name="directory" value="/dmf/spool"/>
            <nvpair name="fstype" value="xfs"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="15s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="10s" on_fail="fence"/>
        <op name="monitor" interval="10s" timeout="10s"  on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
<primitive id="journalsFS" class="ocf" provider="heartbeat" type="Filesystem">
    <instance_attributes>
        <attributes>
            <nvpair name="device" value="/dev/lxvm/journals"/>
            <nvpair name="directory" value="/dmf/journals"/>
            <nvpair name="fstype" value="xfs"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="15s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="10s" on_fail="fence"/>
        <op name="monitor" interval="10s" timeout="10s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
```

```
<primitive id="tmpFS" class="ocf" provider="heartbeat" type="Filesystem">
    <instance_attributes>
        <attributes>
            <nvpair name="device" value="/dev/lxvm/tmp"/>
            <nvpair name="directory" value="/dmf/tmp"/>
            <nvpair name="fstype" value="xfs"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="15s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="10s" on_fail="fence"/>
        <op name="monitor" interval="10s" timeout="10s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
<primitive id="moveFS" class="ocf" provider="heartbeat" type="Filesystem">
    <instance_attributes>
        <attributes>
            <nvpair name="device" value="/dev/lxvm/move"/>
            <nvpair name="directory" value="/dmf/move"/>
            <nvpair name="fstype" value="xfs"/>
            <nvpair name="options" value="dmi,mtpt=/dmf/move"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="15s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="10s" on_fail="fence"/>
        <op name="monitor" interval="10s" timeout="10s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
```

```
<primitive id="diskmspFS" class="ocf" provider="heartbeat" type="Filesystem">
    <instance_attributes>
        <attributes>
            <nvpair name="device" value="/dev/lxvm/diskmsp"/>
            <nvpair name="directory" value="/dmf/diskmsp"/>
            <nvpair name="fstype" value="xfs"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="15s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="10s" on_fail="fence"/>
        <op name="monitor" interval="10s" timeout="10s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
<primitive id="dmfusr1FS" class="ocf" provider="heartbeat" type="Filesystem">
    <instance_attributes>
        <attributes>
            <nvpair name="device" value="/dev/lxvm/dmfusr1"/>
            <nvpair name="directory" value="/dmfusr1"/>
            <nvpair name="fstype" value="xfs"/>
            <nvpair name="options" value="dmi,mtpt=/dmfusr1"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="15s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="10s" on_fail="fence"/>
        <op name="monitor" interval="10s" timeout="10s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
```

```
<primitive id="dmfusr3FS" class="ocf" provider="heartbeat" type="Filesystem">
    <instance_attributes>
        <attributes>
            <nvpair name="device" value="/dev/lxvm/dmfusr3"/>
            <nvpair name="directory" value="/dmfusr3"/>
            <nvpair name="fstype" value="xfs"/>
            <nvpair name="options" value="dmi,mtpt=/dmfusr3"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="15s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="10s" on_fail="fence"/>
        <op name="monitor" interval="10s" timeout="10s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
<primitive id="tmfDevGrp" class="ocf" provider="sgi" type="tmf">
    <instance_attributes>
        <attributes>
            <nvpair name="devgrpnames" value="ibm3592,stk9940b"/>
            <nvpair name="mindevsup" value="2,5"/>
            <nvpair name="devtimeout" value="109,118"/>
            <nvpair name="admin_emails" value="admin1,admin2"/>
            <nvpair name="loader_names" value="ibm3494,1700a"/>
            <nvpair name="loader_hosts" value="ibm3494cps,stk9710"/>
            <nvpair name="loader_users" value="root,acsss"/>
            <nvpair name="loader_passwords" value="my!pass,myOtherPass"/>
        </attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="236s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="236s" on_fail="fence"/>
        <op name="monitor" interval="60s" timeout="30s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
```

```
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
<primitive id="dmfVolGrp" class="ocf" provider="sgi" type="dmf">
    <instance_attributes>
        <attributes>
            <nvpair name="only_email_on_failure" value="false"/>
            <nvpair name="admin_email_addresses" value="root,admin1"/>
        <attributes>
    </instance_attributes>
    <operations>
        <op name="start" timeout="30s" prereq="fencing" on_fail="restart"/>
        <op name="stop" timeout="30s" on_fail="fence"/>
        <op name="monitor" interval="60s" timeout="30s" on_fail="restart"/>
    </operations>
    <meta_attributes>
        <attributes>
            <nvpair name="resource_stickiness" value="1"/>
            <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
    </meta_attributes>
</primitive>
</group>
<clone id="stonith-l2network-set">
  <instance_attributes>
    <attributes>
      <nvpair name="clone_max" value="2"/>
      <nvpair name="clone_node_max" value="1"/>
    </attributes>
  </instance_attributes>
  <primitive id="stonith-l2network" class="stonith" type="l2network">
    <instance_attributes>
      <attributes>
        <nvpair name="nodelist" value="node-0;128.162.244.227;;0 node-1;128.162.244.228;;0"/>
      </attributes>
    </instance_attributes>
    <operations>
      <op id="start" name="start" timeout="5s" prereq="nothing" on_fail="restart"/>
      <op id="monitor" name="monitor" timeout="5s" interval="10s" prereq="nothing" on_fail="restart"/>
```

```
    </operations>
  </primitive>
</clone>
```

Complete XML Example for CXFS, DMF, and TMF

Example A-2 shows a fully defined CXFS, DMF, and TMF service, not including cluster-wide parameters.

Example A-2 Complete XML Example for CXFS, DMF, and TMF

```
<resources>
  <group id="example_Resource_Group">
    <primitive id="CXFS_Filesystems" class="ocf" provider="sgi" type="cxfs">
      <instance_attributes>
        <attributes>
          <nvpair name="volnames" value="home,journals,spool,tmp,move,dmfusr1,diskmsp"/>
        </attributes>
      </instance_attributes>
      <operations>
        <op name="start" timeout="600s" on_fail="restart" prereq="fencing"/>
        <op name="stop" timeout="600s" on_fail="fence"/>
        <op name="monitor" timeout="20s" on_fail="restart"/>
      </operations>
      <meta_attributes>
        <attributes>
          <nvpair name="resource_stickiness" value="1"/>
          <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
        </attributes>
      </meta_attributes>
    </primitive>
    <primitive id="TMF_Device_Groups" class="ocf" provider="sgi" type="tmf">
      <instance_attributes>
        <attributes>
          <nvpair name="devgrpnames" value="CART"/>
          <nvpair name="mindevsup" value="3"/>
          <nvpair name="devtimeout" value="118"/>
          <nvpair name="admin_emails" value="admin@company.com"/>
          <nvpair name="loader_names" value="L700A"/>
```

```
        <nvpair name="loader_hosts" value="stk9710"/>
        <nvpair name="loader_users" value="acsss"/>
        <nvpair name="loader_passwords" value="dog3ear"/>
      </attributes>
    </instance_attributes>
    <operations>
      <op name="start" timeout="236s" prereq="fencing" on_fail="restart"/>
      <op name="stop" timeout="236s" on_fail="fence"/>
      <op name="monitor" interval="60s" timeout="30s" on_fail="restart"/>
    </operations>
    <meta_attributes>
      <attributes>
        <nvpair name="resource_stickiness" value="1"/>
        <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
      </attributes>
    </meta_attributes>
  </primitive>
  <primitive class="ocf" type="dmf" provider="sgi" id="DMF_Resource">
    <instance_attributes>
      <attributes>
        <nvpair name="only_email_on_failure" value="false"/>
        <nvpair name="admin_email_addresses" value="root@localhost,admin1@company.com"/>
      </attributes>
    </instance_attributes>
    <operations>
      <op name="start" timeout="30s" prereq="fencing" on_fail="restart"/>
      <op name="stop" timeout="30s" on_fail="fence"/>
      <op name="monitor" timeout="30s" interval="60s" on_fail="restart"/>
    </operations>
    <meta_attributes>
      <attributes>
        <nvpair name="resource_stickiness" value="1"/>
        <nvpair name="resource_failure_stickiness" value="-INFINITY"/>
      </attributes>
    </meta_attributes>
  </primitive>
</instance_attributes>
<clone>
  <instance_attributes>
    <attributes>
      <nvpair name="clone_max" value="2"/>
```

```
        <nvpair name="clone_node_max" value="1"/>
      </attributes>
    </instance_attributes>
    <primitive id="stonith-l2network" class="stonith" type="l2network">
      <instance_attributes>
        <attributes>
          <nvpair name="nodelist" value="node1;128.162.244.227;;0 node2;128.162.244.228;;0"/>
        </attributes>
      </instance_attributes>
      <operations>
        <op name="start" timeout="5s" prereq="nothing" on_fail="restart"/>
        <op name="monitor" timeout="5s" interval="10s" prereq="nothing" on_fail="restart"/>
      </operations>
    </primitive>
  <clone>
</resources>
```

Differences Among Heartbeat, FailSafe, and SGI Cluster Manager

Table B-1 summarizes the differences among the following, for those readers who may be familiar with with the older products:

- FailSafe
- SGI Cluster Manager
- Linux-HA Heartbeat

Note: These products do not work together and cannot form an HA cluster.

For details about Heartbeat characteristics, see the Linux-HA documentation referred to in "Sources for Detailed Heartbeat Documentation" on page 5.

Table B-1 Differences Among FailSafe, SGI Cluster Manager, and Heartbeat

Topic	FailSafe	SGI Cluster Manager	Heartbeat
Operating system	IRIX	SGI ProPack 5 for Linux	Can be built and run on most operating systems based on UNIX. The version of Heartbeat packaged by SGI is part of the ISSP media distribution and runs on the base OS for ISSP as defined in the ISSP release notes.
Terminology	node resource	member application	node resource
Size of cluster	8 nodes	2 members	8+ nodes (Specific resource agents may have cluster size limitations. DMF can run on only 2 nodes in active/passive mode.)
Node/member name	Hostname or private network address	Hostname only	Hostname and private network address
NFS lock failover	Supported	Not supported	Not supported by the operating system
Network tiebreaker	A node that is participating the cluster membership. FailSafe tries to include the tiebreaker node in the membership in case of a split cluster.	The IP address of machine or a router that **does not participate** in the cluster membership. Usually it is the IP address of a network router that connects the SGI Cluster Manager members to the external world (clients). In a split cluster, only those members that can contact the tiebreaker IP address can form a cluster. There is also a disk tiebreaker.	You can configure Heartbeat to use a variety of methods to provide tiebreaker functionality.
Rolling upgrade	Supported	Not supported	Supported

Topic	FailSafe	SGI Cluster Manager	Heartbeat
Configuration information storage	Information is stored in the cluster database. The cluster database is replicated on all nodes automatically and kept in synchronization.	Information is stored in the `/etc/cluster.xml` configuration file and in the shared partitions. For initial configuration, you must copy this file to all members, such as by using `scp`. After making configuration changes, you must verify that configuration files are in synchronization.	The `/etc/ha.d/ha.cf` file contains bootstrap information and must be manually replicated across the cluster when changed. Other cluster configuration is stored in the cluster information base (CIB), which is a replicated database. You can use `cibadmin`(8) to query and update the CIB.
Making changes while the service is enabled	Depends upon the plug-in and the configuration device parameter.	Device parameter, IP address parameters, and check interval cannot be changed.	Service parameters can be changed while a service is running. Depending on the service and parameter, a change may cause a `stop`/`start` or a trigger a `restart` action. SGI recommends that you do not make any changes that could stop or restart DMF and CXFS.
Heartbeat interval and timeout	You can specify cluster membership heartbeat interval and timeout (in milliseconds).	In the command line, you can specify the heartbeat interval (in microseconds) and the number of heartbeats that can be consecutively missed (`tko_count`). You can also specify the aggregate failover speed in the GUI.	Heartbeat provides a number of parameters to tune node status monitoring and failure actions.
Heartbeat networks	Allows multiple networks to be designated as heartbeat networks. You can choose a list of networks.	Allows heartbeat on all networks or as a multicast on the hostname network. However, you cannot choose a list of networks.	You can configure Heartbeat to communicate over one or more private or public networks.

Topic	FailSafe	SGI Cluster Manager	Heartbeat
Action scripts	Separate scripts named `start`, `stop`, `monitor`, `restart`, `exclusive`.	A `bash` script that contains `start`, `stop`, and `status` parameters. The equivalent for `restart` in SGI Cluster Manager is to perform a `stop` and then a `start`; there is no equivalent in SGI Cluster Manager for `exclusive`.	The Open Cluster Framework (OCF) resource agent specification, which may support `start`, `monitor`, `stop`, and `restart` actions as well as other more-specialized actions.
Resource timeouts	Timeouts can be specified for each action (`start`, `stop`, `monitor`, `restart`, `exclusive`) and for each resource type independently.	Timeout can be specified for each service irrespective of the action or the number of resources it contains.	Timeouts and failover actions are highly configurable.
Resource dependencies	Resource and resource type dependencies are supported and can be modified by the user.	Applications have fixed dependencies. The start and stop order of applications cannot be modified.	Heartbeat provides great flexibility to configure resource dependencies.
Failover policies	The ordered and round-robin failover policies are predefined. User-defined failover policies are supported.	Only the predefined ordered policy is supported. No user-defined failover policies are supported.	Heartbeat provides great flexibility to configure resource failover policies.

Glossary

This glossary lists terms and abbreviations used within this guide. For a more terms, see the documentation referred to in "Sources for Detailed Heartbeat Documentation" on page 5.

active/passive mode

A Heartbeat cluster in which all of the resources run on one node and another node is the standby in case the first node fails.

alternate node

Another node in the Heartbeat cluster other than the first node from which you perform configuration steps as defined in this guide. See also *initial node*.

BMC

Baseboard management controller, a system controller used in resetting Altix XE x86_64 systems.

CIB

Cluster information base, used to define the Heartbeat cluster.

CXFS

Clustered XFS.

DCP

Drive control programs.

DMF

Data Migration Facility, a hierarchical storage management system for SGI environments.

fencing

The method that Heartbeat uses to guarantee a known cluster state when communication to a node fails or actions on a node fail. (This differs from the concept of *fencing* in CXFS.)

HA

Highly available or *high-availability*, in which resources fail over from one node to another without disrupting services for clients.

Heartbeat

The product provided by the Linux-HA Project for high availability.

Heartbeat fail policy

A parameter defined in the CIB that determines what happens when a resource fails.

Heartbeat-managed filesystem

A filesystem that will be made highly available according to the instructions in this guide.

initial node

One host (which will later become a node in the Heartbeat cluster) on which all of the filesystems will be mounted and on which all tape drives and libraries are accessible. See also *alternate node*.

IPMI

Intelligent Platform Management Interface, a system reset method for Altix XE x86_64 systems.

ISSP

SGI *InfiniteStorage Software Platform* is a software distribution.

LCP

Library control programs.

LSB
Linux Standard Base.

OCF
Open Cluster Framework.

OpenVault
A tape mounting service used by DMF.

physvol
XVM physical volume.

primitive
XML code used to define a resource in the CIB.

resource
A service, associated with an IP address, that is managed by Heartbeat.

resource agent
The software that allows a service to be highly available without modifying the application itself.

resource group
A set of resources that are collocated on the same node and ordered to start and stop serially. The resources in a resource group will fail over together as a set.

resource stickiness
A concept in Heartbeat that determines whether a resource should migrate to another node or stay on the node on which it is currently running.

serverdir directory
A directory dedicated to holding OpenVault's database and logs within an HA filesystem in the DMF resource group.

split cluster

A situation in which cluster membership divides into multiple clusters, each claiming ownership of the same filesystems, which can result in filesystem data corruption. Also known as *split-brain syndrome*.

STONITH

Shoot the other node in the head, the facility that guarantees cluster state by fencing non-responsive or failing nodes.

TMF

Tape Management Facility, a tape mounting service used by DMF.

XFS

A filesystem implementation type for the Linux operating system. It defines the format that is used to store data on disks managed by the filesystem.

XML

Extensible mark-up language, used to define resources in the CIB.

XVM

Volume manager for XFS filesystems (local XVM).

Index

P

R