IRIS FailSafeTM Programmer's Guide

Document Number 007-3298-001

CONTRIBUTORS

Written by Susan Ellis
Illustrated by Dany Galgani
Edited by Christina Cary
Production by Ruth Christian
Engineering contributions by Gilberto Arnaiz, Ashwinee Dinkar, Michael
Nishimoto, and Paddy Sreenivasan
Cover design and illustration by Rob Aguilar, Rikk Carey, Dean Hodgkinson,
Erik Lindholm, and Kay Maitz

© 1996, Silicon Graphics, Inc.— All Rights Reserved

The contents of this document may not be copied or duplicated in any form, in whole or in part, without the prior written permission of Silicon Graphics, Inc.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure of the technical data contained in this document by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 and/or in similar or successor clauses in the FAR, or in the DOD or NASA FAR Supplement. Unpublished rights reserved under the Copyright Laws of the United States. Contractor/manufacturer is Silicon Graphics, Inc., 2011 N. Shoreline Blvd., Mountain View, CA 94043-1389.

Silicon Graphics, CHALLENGE, IRIS, and the Silicon Graphics logo are registered trademarks and IRIX and IRIS FailSafe are trademarks of Silicon Graphics, Inc. NFS is a registered trademark of Sun Microsystems, Inc. Netscape is a trademark of Netscape Communications Corporation. Sybase is a registered trademark of Sybase, Inc. Oracle is a registered trademark of Oracle Corporation. INFORMIX is a registered trademark of Informix Software, Inc.

IRIS FailSafe™ Programmer's Guide Document Number 007-3298-001

Contents

List of Figures vii

About This Guide xi

List of Tables ix

1.

Audience xi	
Structure of This Document xii	
Related Documentation xii	
Conventions Used in This Guide xiv	
Introduction to IRIS FailSafe Programming 1	
What Applications Can Be Highly Available Services in an IRIS FailSafe Cluster?	2
IRIS FailSafe Programming Steps 3	
IRIS FailSafe System Software 4	
Software Components 4	
Node States 7	
Failover Operations 8	
State Transition Scenarios 9	
Concurrent Reboot 11	
Reboot of a Node While the Other Node Is Down or Not Running IRIS FailSafe 12	
System Maintenance Shutdown 13	
Restart After System Maintenance (No Reboot) 14	
Restart After System Maintenance With Reboot 15	
Node Failure 16	
Recovery After Node Failure With the Controlled-Failback Parameter Set to False (or Not Set) 17	
Recovery After Node Failure With the Controlled-Failback Parameter Set	18
Node in Normal State Detects a Local Failure 19	

2.

3.

4.

Preparing to Write a Failover Script 54

Understanding the Failover Script Template 56

Standby State 20 Node in Degraded State Detects Local Failure While the Other Node Is in Controlled Failback State 21 Node in Controlled Failback State Detects the Failure of a Node in Degraded State 22 IRIS FailSafe Scripts 22 Tasks Performed by the Standard Failover Scripts 26 Filesystems 26 Interfaces 27 NFS 27 Statd 28 Volumes 28 Webserver 29 Modifying the Configuration File for a New Highly Available Service 31 An Example of Configuration File Information for a New Highly Available Service 31 Adding a New Highly Available Service to the Configuration File 33 Choosing Parameters for a New Highly Available Service 34 Checking the Configuration File 35 Writing a Monitoring Script 37 Writing a Monitoring Script 37 Preparing to Write a Monitoring Script 39 Understanding the Monitoring Script Template 41 Defining Variables for New Block, Section, and Parameter Names 45 Using ha_cfginfo to Get Configuration File Information 46 Understanding the Function of the Monitoring Script check() Function 48 Executing a Command in a Monitoring Script 50 Writing a Failover Script 53 Writing a Failover Script 53

Node in Degraded State Detects Local Failure While the Other Node Is in

Writing the Failover Functions 62
Executing a Command in a Failover Script 64

5. Installing and Testing Scripts 65

Naming and Installing Monitoring Scripts 65

Choosing the Execution Order of Failover Scripts for Each Operation 66

Installing Failover Scripts 67

Modifying Application Startup Procedures 67

Testing New Scripts 68

General Testing and Debugging Techniques 68

Testing Monitoring Scripts 69

Testing Failover Scripts Without Starting IRIS FailSafe 70

Testing Failover Scripts While Running IRIS FailSafe 72

A. Names Used in Template Configuration Files 75

Block Names 76

Section Names 77

Parameter Names 78

List of Figures

Figure 1-1	Message Paths Between Software Components While Nodes Are in Normal State 6
Figure 1-2	Concurrent Reboot 11
Figure 1-3	Reboot of a Node While the Other Node Is Down or Not Running IRIS FailSafe 12
Figure 1-4	System Maintenance Shutdown 13
Figure 1-5	Restart After System Maintenance (No Reboot) 14
Figure 1-6	System Maintenance Restart With Reboot 15
Figure 1-7	Node Failure 16
Figure 1-8	Recovery After Node Failure With the Controlled-Failback Parameter Set to False (or Not Set) 17
Figure 1-9	Recovery After Node Failure With the Controlled-Failback Parameter Set 18
Figure 1-10	Node in Normal State Detects a Local Failure 19
Figure 1-11	Node in Degraded State Detects Local Failure While the Other Node Is in Standby State 20
Figure 1-12	Node in Degraded State Detects Local Failure While the Other Node Is in Controlled Failback State 21
Figure 1-13	Node in Controlled Failback State Detects the Failure of a Node in Degraded State 22
Figure 1-14	Directory Structure of /var/ha 23

List of Tables

Table 1-1	IRIS FailSafe Software 4
Table 1-2	IRIS FailSafe Administrative Commands 7
Table A-1	Major Blocks in the Configuration File 76
Table A-2	Section Names in Template Configuration Files 77
Table A-3	Parameters in Template Configuration Files 78

About This Guide

The IRIS FailSafe [™] system is a pair of servers (called *nodes* in this guide) that communicate with each other using serial cables and a private Ethernet connection in addition to using the public network. The nodes may be configured in a dual-hosted or dual-initiator configuration. IRIS FailSafe software enables the nodes to be configured so that if one node fails, applications and services running on the failed node can be transferred (*failed over*) to the remaining node. IRIS FailSafe software and IRIS FailSafe software options fail over specific applications and resources, called *highly available services*, such as databases, Web servers, NFS[®] filesystems, XLV logical volumes, and IP addresses. This guide explains how to write the set of scripts that are required to turn an application into a highly available service.

This guide assumes that the IRIS FailSafe system has been set up and configured as described in the *IRIS FailSafe Administrator's Guide* and is able to successfully fail over highly available services. This guide requires that the configuration file /var/ha/ha.conf be an IRIS FailSafe Release 1.1 configuration file (version-major = 1 and version-minor = 1 in /var/ha/ha.conf).

This guide was prepared in conjunction with Release 1.1 of IRIS FailSafe.

Audience

This guide is written for system programmers who are developing scripts for the IRIS FailSafe system that enable it to fail over applications that aren't handled by the base and optional IRIS FailSafe products. These programmers must be familiar with the operation and administration of nodes running IRIS FailSafe, with the applications that are to be failed over, and with the *IRIS FailSafe Administrator's Guide*.

Structure of This Document

This guide contains the following chapters and appendix:

- Chapter 1, "Introduction to IRIS FailSafe Programming," introduces the software components of the IRIS FailSafe system and provides an overview on the steps required to add a new, highly available service to the system.
- Chapter 2, "Modifying the Configuration File for a New Highly Available Service," explains how to add configuration information to the IRIS FailSafe configuration file
- Chapter 3, "Writing a Monitoring Script," describes how to write local and remote monitoring scripts.
- Chapter 4, "Writing a Failover Script," explains how to write a failover script for a highly available service.
- Chapter 5, "Installing and Testing Scripts," describes how to test the newly added scripts.
- Appendix A, "Names Used in Template Configuration Files," lists the names used in the configuration file and their purposes.

Related Documentation

Besides this guide, other documentation for the IRIS FailSafe system includes

- IRIS FailSafe Administrator's Guide
- IRIS FailSafe Sybase Administrator's Guide (IRIS FailSafe Sybase® option)
- IRIS FailSafe Oracle Administrator's Guide (IRIS FailSafe Oracle® option)
- IRIS FailSafe INFORMIX Administrator's Guide (IRIS FailSafe INFORMIX® option)

The IRIS FailSafe reference pages are as follows:

- ha_admin(1M)
- ha_appmon(1M)
- ha_cfgchksum(1M)
- ha_cfginfo(1M)
- ha_cfgverify(1M)

- ha_exec(1M)
- ha_hbeat(1M)
- ha_ifa(1M)
- ha_ifmx(1M) (IRIS FailSafe INFORMIX option)
- ha_killd(1M)
- ha_nc(1M)
- ha_orcl(1M) (IRIS FailSafe Oracle option)
- ha_spng(1M)
- ha_sybs(1M) (IRIS FailSafe Sybase option)
- http_ping(1M) (IRIS FailSafe Web option)
- macconfig(1M)
- ha.conf(4)
- failsafe(7M)

Release notes are included with each IRIS FailSafe product. The names of the release notes are as follows:

ha_base release notes for IRIS FailSafe
ha_nfs release notes for IRIS FailSafe NFS
ha_www release notes for IRIS FailSafe Web
ha_ orcl release notes for IRIS FailSafe Oracle
ha_ ifmx release notes for IRIS FailSafe INFORMIX
ha_ sybs release notes for IRIS FailSafe Sybase

Conventions Used in This Guide

These type conventions and symbols are used in this guide:

Bold Literal command-line arguments and literal parameter values

Italics Command names, filenames, new terms, the names of inst subsystems,

manual/book titles, variable command-line arguments, and variables to be supplied by the user in examples, code, and syntax statements

Fixed-width type

Examples of command output that is displayed in windows on your

monitor and of the contents of files

Bold fixed-width type

 $Commands \ and \ text \ that \ you \ are \ to \ type \ literally \ in \ response \ to \ shell \ and$

command prompts

IRIXTM shell prompt for the superuser (root)

Introduction to IRIS FailSafe Programming

Programmers must write several scripts and add information to the IRIS FailSafe configuration file to turn an application into a highly available service supported by the IRIS FailSafe system. No modifications need to be made to an application to make it a highly available service—all changes are to the IRIS FailSafe system software and to the IRIS FailSafe configuration file.

Not every application can become a highly available service—this chapter explains the characteristics an application must have to become a highly available service. This chapter also describes the procedure for developing the scripts that make an application a highly available service and gives information about IRIS FailSafe system software that is important for programmers developing these scripts.

The major sections in this chapter are as follows:

- "What Applications Can Be Highly Available Services in an IRIS FailSafe Cluster?" on page 2
- "IRIS FailSafe Programming Steps" on page 3
- "IRIS FailSafe System Software" on page 4

What Applications Can Be Highly Available Services in an IRIS FailSafe Cluster?

Applications need not be modified to make them highly available when running in an FailSafe cluster. However, not every application is appropriate for high availability. This section describes the characteristics that an application must have for high availability. When an application has these characteristics, a programmer can develop the scripts required by the IRIS FailSafe system.

The characteristics of an application that can be made highly available are:

- The application is crash tolerant.
 - It should be able to recover from failures. The failure could be a system failure (a hardware failure or an operating system failure) or an application failure.
- The application must have a start and stop procedure.
 - When the application fails over, the instances of the application are stopped on one node using the stop procedure and restarted on the other node using start procedure.
- The application can be moved from one node to the other after failures.
 - If the application has failed, it must still be possible to run the application stop procedure. In addition, the application must recover from the failed state when the application start procedure is executed in the other node.
- If the application depends on other applications and resources, they must also be highly available.
 - IRIS FailSafe and its optional products enable you to configure system resources (interfaces, IP addresses, XLV volumes, and XFS filesystems) and applications (NFS filesystems, Netscape servers, and INFORMIX, Oracle, and Sybase databases) as highly available services. For these highly available services, you must configure them as described in the *IRIS FailSafe Administrator's Guide*.

If the application you want to make highly available depends on resources and applications not supported by IRIS FailSafe products, you must make these resources and applications highly available, too, using the information in this guide.

IRIS FailSafe Programming Steps

To turn an application into a highly available service automatically failed over by the IRIS FailSafe system, follow these steps:

- 1. Understand the application and determine:
 - The system resources on which the application depends.
 - The type of configuration required for the applications—active/backup or dual-active.
 - The number of instances of the application.
 - The commands and arguments required to start, stop, and monitor the application instances.
 - The relationships between this application and other highly available services; specifically, the order all highly available services need to be started and stopped.
- 2. Configure and test the base IRIS FailSafe system as described in the *IRIS FailSafe Administrator's Guide*.
- 3. Add configuration information for the new application to /var/ha/ha.conf. See Chapter 2, "Modifying the Configuration File for a New Highly Available Service."
- 4. Write the local and remote monitoring scripts. See Chapter 3, "Writing a Monitoring Script."
- 5. Write the failover script. This script includes the **giveaway()**, **giveback()**, **takeback()**, and **takeover()** functions. See Chapter 4, "Writing a Failover Script."
- 6. Install, test, and debug the scripts. See Chapter 5, "Installing and Testing Scripts."

Note: Do not modify the scripts included with the IRIS FailSafe product. New or customized scripts must have different names than the files included with IRIS FailSafe (see Figure 1-14).

IRIS FailSafe System Software

In this section, the architecture of the system software is described: the software components and the messages they send, the states that nodes can be in and how they transition to other states, and the scripts that enable fail over of highly available services. This material supplements the overview IRIS FailSafe system provided in Chapter 1 of the IRIS FailSafe Administrator's Guide.

Software Components

Table 1-1 describes the IRIS FailSafe system software executables.

 Table 1-1
 IRIS FailSafe Software

Component	Process	Purpose
Heartbeat daemon	/usr/etc/ha_hbeat	Generates heartbeats, which are messages passed between the two nodes of the cluster over the private Ethernet that connects the nodes.
Node controller	/usr/etc/ha_nc	Keeps track of the node state. Determines when a state transition is needed as a result of a heartbeat failure, an application failure, an interface failure, an administrative command, or a node re-entering a cluster. Controls actions such as killing the other node; leaving, rejoining, and reintegrating into a cluster; and starting and stopping monitoring.
Application monitor	/usr/etc/ha_appmon	Monitors the node's highly available services. Monitors the remote heartbeat daemon and local node controller. Notifies the node controller of failures. Executes scripts. Sends mail at state transitions and failures.
Kill daemon	/usr/etc/ha_killd	Monitors the serial connection to the other node. Has the ability to kill the other node.
Local monitoring scripts	/var/ha/actions/ha_*_lmon	Each local monitoring script monitors all instances of one highly available service on the local node.
Database agents	/usr/etc/ha_ifmx, /usr/etc/ha_orcl, /usr/etc/ha_sybs	Each database agent monitors all instances of one type of database.

 Table 1-1 (continued)
 IRIS FailSafe Software

Component	Process	Purpose
Interface agent	/usr/etc/ha_ifa	Monitors the local node's interfaces.
Failover scripts for highly available services	/var/ha/resources/*	Each script contains four failover functions (takeback(), takeover(), giveaway(), and giveback()) for one highly available service.
Failover scripts for operations	/var/ha/actions/{giveaway, giveback,takeback,takeover}	Each of these scripts causes the failover scripts in /var/ha/resources to be executed with the name of the calling script as a command-line argument. This causes the function with the same name as the argument (takeback(), takeover(), giveaway(), or giveback()) to be executed.

Figure 1-1 shows the communication paths between the software components listed in Table 1-1 on the two nodes of an IRIS FailSafe cluster while they are in normal state. All of the processes shown run continuously during normal IRIS FailSafe operation except the monitoring scripts and failover scripts. The monitoring scripts run periodically to check the state of agents and applications. The failover scripts run only during fail over.

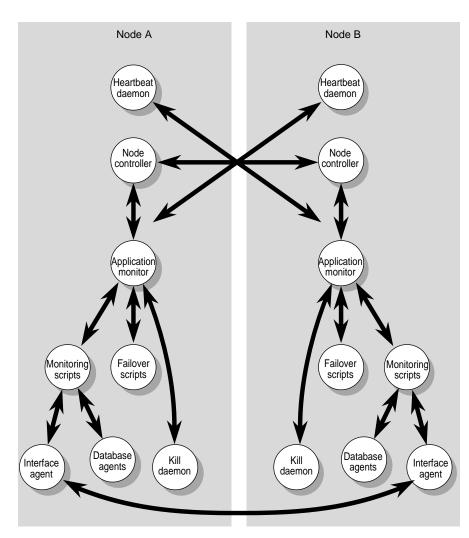


Figure 1-1 Message Paths Between Software Components While Nodes Are in Normal State

Table 1-2 describes the IRIS FailSafe administrative commands.

 Table 1-2
 IRIS FailSafe Administrative Commands

Command Purpose	
ha_admin	Enable the system administrator to force IRIS FailSafe failover operations.
ha_cfgchksum	Compute the checksum of a configuration file.
ha_cfginfo	Extract information from the configuration file.
ha_cfgverify	Verify a configuration file.
ha_exec	Execute a command, possibly repeatedly.
ha_spng	Verify the serial connections between nodes.
http_ping	Check if a Netscape $^{\text{\tiny TM}}$ server is still running.
macconfig	Display or modify MAC addresses.

The IRIS FailSafe configuration file /var/ha/ha.conf is a key component of IRIS FailSafe software. It contains cluster-specific configuration information, which is used by IRIS FailSafe monitoring and failover scripts. The configuration files on the nodes in a cluster must be identical.

Node States

The condition of each node in a cluster is described by its *state*. The possible states are as follows:

standalone	The node is coming up and IRIS FailSafe is starting up. This is a transient state.
joining	The node is coming up and joining the cluster. The node should never remain in this state for more than two or three minutes.
normal	The node is actively providing its own highly available services. It is also monitoring the other node in the cluster.
degraded	The node is providing all highly available services for the cluster; the other node is unavailable.

standby This node has stopped monitoring the other node in the cluster and is no

longer providing highly available services. It is not capable of providing

highly available services.

controlled failback

This node is no longer providing highly available services, but it is monitoring the other node in the cluster and the services it is providing. The node is capable of providing highly available services if the other

node fails.

error An unrecoverable failure has occurred.

Evaluating the current node state and monitoring the state of the other node in the cluster are the responsibilities of the IRIS FailSafe node controller (*ha_nc*). The state of each node in the cluster is kept by each node controller.

Failover Operations

Nodes move from one state to another as a result of system bootup, system halt, failure (system failure or the failure of a highly available service), and *ha_admin* commands. To move from one state to another, a node controller sends a message to its local application monitor and asks it to perform one or more of these four operations, called failover operations:

takeback Start up all services for which this node is listed as the primary node

(specified by the server-node parameter for the service) in ha.conf.

takeover Start up all services for which this node is listed as the backup node

(specified by the backup-node parameter for the service) in *ha.conf*.

giveaway Stop all services for which this node is listed as the primary node for the

service in ha.conf.

giveback Stop all services for which this node is listed as the backup node for the

service in ha.conf.

State Transition Scenarios

The possible state transition scenarios are shown in Figure 1-2 through Figure 1-13 in the following subsections. They show the sequences of states for the two nodes in a cluster, the primary messages passed between the nodes, and the takeback, takeover, giveaway, and giveback operations that occur during the different scenarios. Understanding these scenarios can help you interpret messages written to logs and to debug the scripts you write when you add the failover of a new service to the IRIS FailSafe system.

Figure 1-2 through Figure 1-13 show two nodes over time (time increases from the top of the figures to the bottom). The components of the figures are:

NORMAL, CONTROLLED FAILBACK, DEGRADED, JOINING, STANDALONE,

STANDBY

These names and their colored blocks show the current state of a node. See the section "Node States" in this chapter for descriptions of the states.

diagonal lines

A block of unlabeled diagonal lines indicates that the node is down or it is up, but IRIS FailSafe is not running.

ovals

Ovals are used to indicate events such as crashes, reboots, and the

detection of failures by IRIS FailSafe software.

takeback, takeover, giveaway, giveback

These names indicate that this failover operation is performed. See the section "Failover Operations" in this chapter for descriptions of the failover operations.

ha_admin commands

ha_admin commands entered by system administrators are shown next to the node on which they are entered, for example, # ha_admin -fr.

arrows

Arrows are used with message names to indicate messages from one node to the other. The remainder of this list describes these messages.

kill

A kill message from one node to the other causes the recipient to shut down.

rejoin

A message sent when a node is in joining state. It is sent when a node comes up initially. If there is no response from the other node, the node sending the message cannot move to normal state. Additional rejoin messages are sent every 30 seconds until long-timeout seconds have passed, in case the initial messages are lost. The diagrams show only the one rejoin message.

reintegrate A message similar to rejoin, except that it is sent after the ha_admin -fr

command or *ha_admin -G* command is executed on a node in standby state. It requests permission for the node to move to normal state or

degraded state, respectively.

released A message sent after the *giveback* script has been executed or as a

response to a rejoin or reintegrate message.

standby A message that tells a node to stay in standby state.

start_rmon A message from a node in degraded state that tells the other node to

begin remote monitoring. Normally, a node in standby state doesn't do heartbeat or serial line monitoring of the node in degraded state. This message, sent when the parameter controlled-failback is **true** for the other node, tells the node in standby state to begin monitoring. The other node in standby state moves to controlled failback state.

takeover A message from one node that tells the other node to perform the

takeover operation.

The timeout event indicates that a node does not respond to the previous

message.

Concurrent Reboot

In Figure 1-2, two nodes are rebooted or IRIS FailSafe is started on both nodes within the value of long-timeout seconds (specified in *ha.conf*) of each other. Each node moves to normal state.

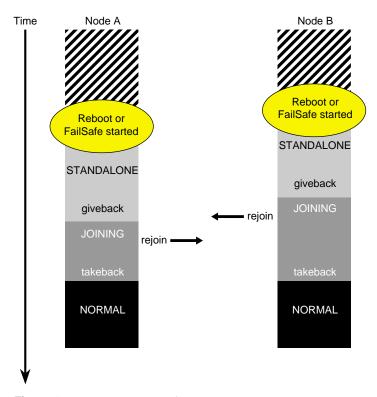


Figure 1-2 Concurrent Reboot

Reboot of a Node While the Other Node Is Down or Not Running IRIS FailSafe

In the scenario shown in Figure 1-3, one node (Node B) is down or not running IRIS FailSafe. The other node (Node A) is rebooted or IRIS FailSafe is started. Node A moves to degraded state.

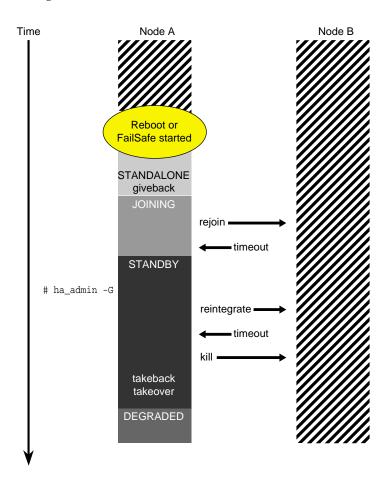


Figure 1-3 Reboot of a Node While the Other Node Is Down or Not Running IRIS FailSafe

System Maintenance Shutdown

In the scenario shown in Figure 1-4, both nodes begin in normal state. The *ha_admin -s* command is entered by the system administrator on one node (node B) to cause all highly available services to fail over to the other node (Node A) in preparation for performing system maintenance on Node B.

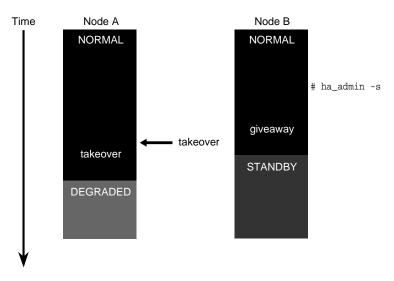


Figure 1-4 System Maintenance Shutdown

Restart After System Maintenance (No Reboot)

In the scenario shown in Figure 1-5, one node (Node A) has been providing all highly available services while system maintenance has been performed on the other node (Node B). This system maintenance doesn't include a reboot. The *ha_admin -fr* command is entered by the system administrator on Node B after system maintenance is completed. Both nodes move automatically to normal state.

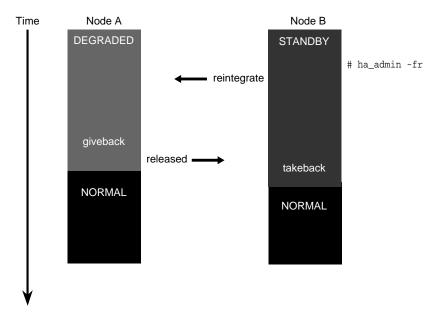


Figure 1-5 Restart After System Maintenance (No Reboot)

Restart After System Maintenance With Reboot

In the scenario shown in Figure 1-6, one node (Node A) has been providing all highly available services while system maintenance has been performed on the other node (Node B). (Node B was brought to standby state using the command *ha_admin -s*.) The system maintenance included a reboot. After the reboot, Node B moves to standby state after attempting to rejoin. The command *ha_admin -fr* must be used to move both nodes to normal state.

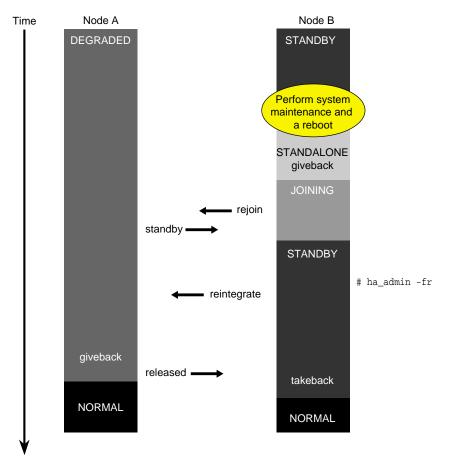


Figure 1-6 System Maintenance Restart With Reboot

Node Failure

In the scenario shown in Figure 1-7, both nodes are operating normally when one node (Node B) crashes or hangs (shown in Figure 1-7). The other node (Node A) moves to degraded state.

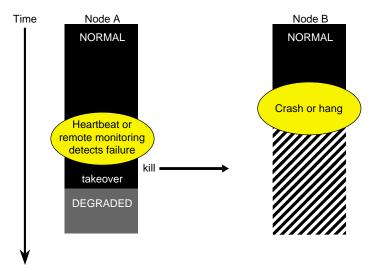


Figure 1-7 Node Failure

Recovery After Node Failure With the Controlled-Failback Parameter Set to False (or Not Set)

In the scenario shown in Figure 1-8, one node (Node A) is operating in degraded state and the other node (Node B) is down or not running IRIS FailSafe. Node B is in an indeterminate state as a result of the scenario in Figure 1-7. It got to When Node B is rebooted, both nodes are brought automatically to normal state because the configuration parameter controlled failback is set to **false** for Node B (or not set for Node B).

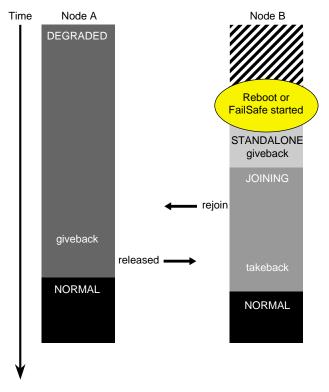


Figure 1-8 Recovery After Node Failure With the Controlled-Failback Parameter Set to False (or Not Set)

Recovery After Node Failure With the Controlled-Failback Parameter Set

In the scenario shown in Figure 1-9, one node (Node A) is operating in degraded state and the other node (Node B) is down or is not running IRIS FailSafe because it has been killed (by Node A or by an incorrect shutdown of IRIS FailSafe). Node B is rebooted, but because the controlled-failback parameter is set to **true** for Node B, Node B goes to controlled failback state. After the *ha_admin-fr* command is entered by the system administrator, both nodes move to normal state.

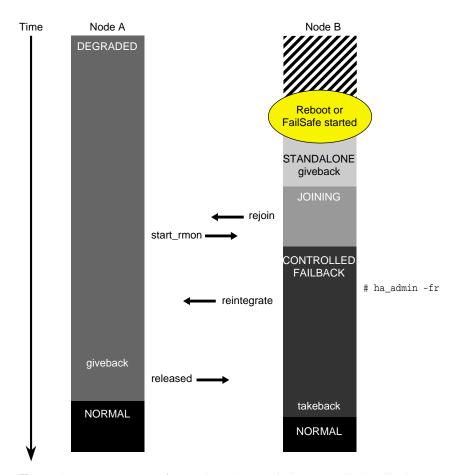


Figure 1-9 Recovery After Node Failure With the Controlled-Failback Parameter Set

Node in Normal State Detects a Local Failure

In the scenario shown in Figure 1-10, both nodes begin in normal state. One node (Node A) detects a local failure and moves to standby state. The other node (Node B) takes over all services and moves to degraded state.

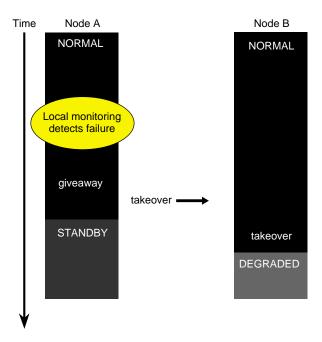


Figure 1-10 Node in Normal State Detects a Local Failure

Node in Degraded State Detects Local Failure While the Other Node Is in Standby State

In the scenario shown in Figure 1-11, one node (Node A) is in degraded state and the other node (Node B) is in standby state. Local monitoring on Node A detects that there has been a failure of a highly available service. All highly available services remain on Node A. Neither node changes state. Local monitoring on Node A is stopped.

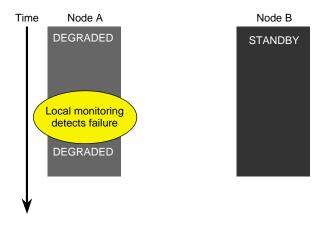


Figure 1-11 Node in Degraded State Detects Local Failure While the Other Node Is in Standby State

Node in Degraded State Detects Local Failure While the Other Node Is in Controlled Failback State

In the scenario shown in Figure 1-12, one node (Node A) is in degraded state and the other node (Node B) is in controlled failback state. Node A detects the failure of a highly available service that it is providing. Node A moves to standby state and Node B moves to degraded state. Node B provides all highly available services.

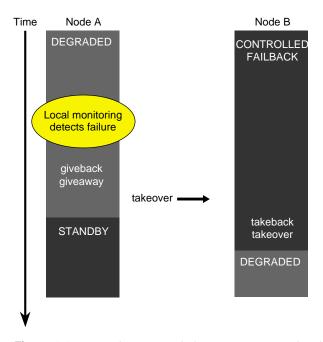


Figure 1-12 Node in Degraded State Detects Local Failure While the Other Node Is in Controlled Failback State

Node in Controlled Failback State Detects the Failure of a Node in Degraded State

In the scenario shown in Figure 1-13, one node (Node A) is in degraded state and the other node (Node B) is in controlled failback state. Heartbeat monitoring on Node B detects a failure on Node A. All highly available services are failed over to node B, which transitions to degraded state. The node that was originally in degraded state is killed and restarted.

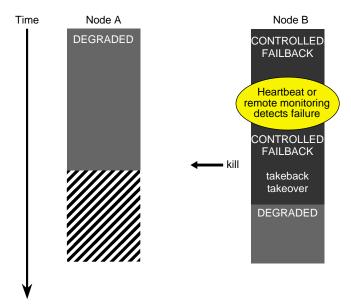


Figure 1-13 Node in Controlled Failback State Detects the Failure of a Node in Degraded State

IRIS FailSafe Scripts

Scripts perform monitoring and the failover operations giveaway, giveback, takeback, and takeover. To add the failover of a new class of applications, you must create monitoring and failover scripts (described in Chapter 3, "Writing a Monitoring Script," and Chapter 4, "Writing a Failover Script") for the new application class.

IRIS FailSafe scripts and configuration file are stored in the directory /var/ha. Figure 1-14 shows the contents of this directory.

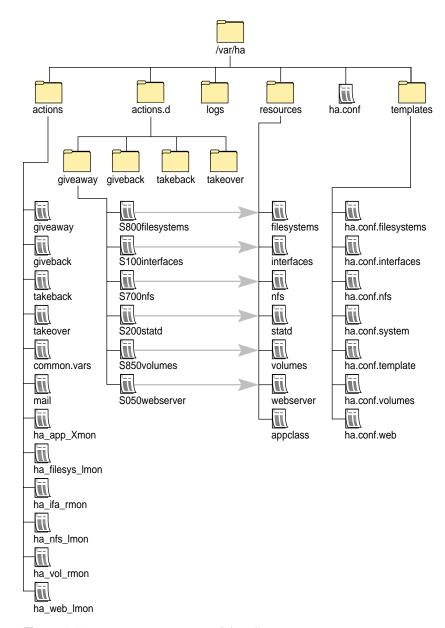


Figure 1-14 Directory Structure of /var/ha

The directory /var/ha contains several categories of files and directories as follows:

actions/giveaway, actions/giveback, actions/takeback, actions/takeover

These scripts execute this operation (giveaway, giveback, takeback, or takeover) for each of the IRIS FailSafe highly available services. They do this by executing all scripts (in lexical order) in the *actions.d* directory (giveaway, giveback, takeback, and takeover) and specifying the operation as a command-line argument.

actions/common.vars

common.vars defines variables used in all of the scripts. Most variables defined in *common.vars* have values that are parameter names. An example is

T_SERVER=server-node

T_SERVER is a variable used in the scripts to get the value of the parameter server-node from the configuration file *ha.conf*.

common.vars is sourced by each IRIS FailSafe monitoring and failover script at the beginning of the script.

actions/mail

This script sends a mail message (using /usr/sbin/Mail) to the address specified by the parameter mail-dest-addr in /var/ha/ha.conf when various types of errors are detected. It is executed by the application monitor (ha_appmon).

actions/ha_<app>_lmon

These scripts are local monitoring scripts. Each script monitors the highly available service with a name similar (or identical) to *<app>*. When you add a highly available service to the IRIS FailSafe system, you must add a local monitoring script. See Chapter 3, "Writing a Monitoring Script," for information about writing a local monitoring script.

logs

This directory contains the logs of all scripts executed by IRIS FailSafe. The outputs and errors from the commands in the scripts are logged.

resources

This directory contains one script per highly available service, plus the script *statd*, which is required by NFS. These scripts contain functions for the IRIS FailSafe operations giveaway, giveback, takeback, and takeover.

resources/<appclass>

Each highly available service added to the IRIS FailSafe system must have a script added to the resources directory that contains **takeback()**, **takeover()**, **giveaway()**, and **giveback()** functions for the highly available service. (See Chapter 4, "Writing a Failover Script," for more details.)

actions.d/giveaway

This directory contains links to each of the files in the *resources* directory. The names of the links are the same as the files in the *resources* directory that have "S" and a three-digit number prepended. Because the files in this directory are executed in lexical order, the three-digit numbers that control the execution order are chosen with care. (See the section "Choosing the Execution Order of Failover Scripts for Each Operation" in Chapter 5 for more details.)

When the files in *actions/giveaway* are executed, they are executed with the command-line argument **giveaway** (the name of the directory). This causes the **giveaway()** function in the script to be performed. For example, executing

S800filesystems checksum giveaway

really executes *resources/filesystems* (because *S800 filesystems* is a link to *resources/filesystems*) with the argument **giveaway**. *checksum* is the checksum for *ha.conf*.

actions.d/giveaway/S<nnn><appclass>

This is a link to the file *resources*/<appclass>. <nnn> is a three-digit number chosen so that this file is executed in the correct order with respect to the other links in this directory. (See the section "Choosing the Execution Order of Failover Scripts for Each Operation" in Chapter 5 for more details.)

actions.d/giveback, actions.d/takeback, actions.d/takeover

These directories are analogous to *actions.d/giveaway*: they contain links to files in the *resources* directory. The links are executed in lexical order with the name of the directory (*giveback*, *takeback*, or *takeover*) as an argument.

ha.conf

This file is the IRIS FailSafe configuration file. It is not included with the IRIS FailSafe product or any of its options; it must be created by the user. (See Chapter 4, "Creating the IRIS FailSafe Configuration File," of the IRIS FailSafe Administrator's Guide for more details.)

templates This directory contains template files used to create the *ha.conf* file. (See

Chapter 4, "Creating the IRIS FailSafe Configuration File," of the IRIS

FailSafe Administrator's Guide for more details.)

Tasks Performed by the Standard Failover Scripts

When the IRIS FailSafe system performs a failover operation (giveaway, giveback, takeback, or takeover), the function with that name in each of the scripts in the /var/ha/resources directory is executed. The order of execution is controlled by the filenames in the /var/ha/actions.d/giveaway, /var/ha/actions.d/giveback, /var/ha/actions.d/takeback, and /var/ha/actions.d/takeover directories as explained in the section "Choosing the Execution Order of Failover Scripts for Each Operation" in Chapter 5. Because all scripts are executed when a failover operation occurs, it is important to understand what tasks are going to be performed by other scripts when writing a new script. For your reference, the tasks performed by each of the standard scripts, the NFS scripts and the Web script for each of the failover operations are described below. The NFS scripts (nfs and statd) and Web script (webserver) are present only if the optional IRIS FailSafe NFS and IRIS FailSafe Web products are installed.

Filesystems

The actions performed by each function in the *filesystems* script are as follows:

takeback() Mounts all filesystems for which this node is the primary node. They are

mounted in increasing lexicographic order.

takeover() Mounts all filesystems for which this node is the backup node. They are

mounted in increasing lexicographic order.

giveaway() Unmounts all filesystems for which this node is the primary node in

decreasing lexicographic order. Before unmounting, it kills any nfsd processes. After unmounting, it restarts all *nfsd* processes.

Note: Any processes with open file descriptors for the unmounted

filesystems are killed.

giveback() Unmounts all filesystems for which this node is the backup node in decreasing lexicographic order. Before unmounting, it kills any nfsd

processes. After unmounting, it restarts all *nfsd* processes.

Note: Any processes with open file descriptors for the unmounted

filesystems are killed.

Interfaces

The actions performed by each function in the *interfaces* script are as follows:

takeback()	Adds all the IP addresses for which this node is the primary node to
	their network interfaces. Also restores the MAC addresses to the
	interfaces, if re-mac'ing is used.

takeover() Adds all the IP addresses for which this node is the backup node to their network interfaces. Also moves the MAC addresses to the interfaces, if re-mac'ing is used.

giveaway() Removes IP addresses from the network interfaces for which this node is the primary node.

giveback() Removes IP addresses from the network interfaces for which this node is the backup node. Restores the MAC addresses if re-mac'ing is used.

NFS

The actions performed by each function in the *nfs* script are as follows:

node.

takeback() Exports filesystems for which this node is the primary node. They are exported in a sorted hierarchical order.
 takeover() Exports filesystems for which this node is the backup node. They are exported in a sorted hierarchical order.
 giveaway() Unexports (exportfs -u) any filesystem for which this node is the primary node.
 giveback() Unexports (exportfs -u) any filesystem for which this node is the backup

Statd

The actions performed by each function in the *statd* script are as follows:

takeback() Add an NFS locking directory (called the *statmon* directory) to be used for all NFS filesystems for which this node is the primary node. Also send out a message instructing NFS clients to reacquire NFS locks if that client had an NFS lock to one of the IP addresses for which this node is

primary node.

takeover() Add an NFS locking directory (called the *statmon* directory) to be used

for all NFS filesystems for which this node is the backup node. Also send out a message instructing NFS clients to reacquire NFS locks if that client had an NFS lock to one of the IP addresses for which this node is backup

node.

giveaway() Remove the NFS locking directory added by **takeback()**.

giveback() Remove the NFS locking directory added by **takeover()**.

Volumes

The actions performed by each function in the *volumes* script are as follows:

takeback() Changes the node names of all of the XLV volumes that are owned by

the backup node to this node's (the primary node) name. Assembles all

the local XLV volumes and starts *xlv_labd* if necessary.

takeover() Changes the node names of all of the XLV volumes that are owned by

the primary node to this node's (the backup node) name. Assembles all

the local XLV volumes and starts *xlv_labd* if necessary.

giveaway() Shuts down all the XLV volumes for which this node is the primary

node. Changes the node name for these XLV volumes to the other node's

name.

giveback() Shuts down all the XLV volumes for which this node is the backup node.

Changes the node name for these XLV volumes to the other node's

name.

Webserver

The actions performed by each function in the *webserver* script are as follows:

takeback()

Starts all Netscape servers for which this node is the primary node. Reads all the webserver blocks and puts all directories specified by httpd-dir for which this is the primary node into the configuration file specified by httpd-options-file. Calls the script specified by httpd-script with the **start** argument.

takeover()

Starts all Netscape servers for which this node is the backup node. Reads all the webserver blocks and puts all directories specified by httpd-dir for which this is the backup node into the configuration file specified by httpd-options-file. Calls the script specified by httpd-script with the **start** argument.

giveaway()

Stops all Netscape servers for which this is the primary node. Reads all the webserver blocks and puts all the directories specified by httpd-dir for which this is the primary node into the configuration file specified by httpd-options-file. Calls the script specified by httpd-script with the **stop** argument.

giveback()

Stops all Netscape servers for which this is the backup node. Reads all the webserver blocks and puts all the directories specified by httpd-dir for which this is the backup node into the configuration file specified by httpd-options-file. Calls the script specified by httpd-script with the **stop** argument.

Modifying the Configuration File for a New Highly Available Service

This chapter describes how to add configuration information about a new service to the IRIS FailSafe configuration file /var/ha/ha.conf. It assumes that you are familiar with creating a configuration file, which is described in Chapter 4 of the IRIS FailSafe Administrator's Guide.

This chapter begins by showing an example: configuration file information for the application *named*, the Internet domain name server. It then describes the process of developing this configuration file text. Other sections provide information about adding new block, section, and parameter names to the configuration file and about checking the syntax of the configuration file.

The sections in this chapter are as follows:

- "An Example of Configuration File Information for a New Highly Available Service" on page 31
- "Adding a New Highly Available Service to the Configuration File" on page 33
- "Choosing Parameters for a New Highly Available Service" on page 34
- "Checking the Configuration File" on page 35

An Example of Configuration File Information for a New Highly Available Service

Like all highly available services supported by IRIS FailSafe software options (interfaces, volumes, filesystems, nfs, and so on), the configuration of the application you are making highly available must be described in the IRIS FailSafe configuration file /var/ha/ha.conf.

For example, if you are adding *named* as a highly available application, the blocks and parameters for *named* in the configuration file might be as follows (shown with line numbers):

```
1 application-class named
 3
                                   # active node's hostname
       server-node = hal
 4 }
 6 action named
 7 {
 8
       local-monitor = /var/ha/actions/ha_named_lmon
 9 }
10
11 action-timer named
12 {
13
       start-monitor-time = 60
14
       lmon-probe-time = 120
15
       lmon-timeout = 30
16 }
17
18 named named_block
19 {
20
       server-node = hal
                                   # primary node's hostname
21
                                   # backup node's hostname
       backup-node = ha2
22
       process-name = named
23 }
```

named is an application that needs one instance running in an IRIS FailSafe cluster. Thus, it is run in an active/backup configuration and just one server-node parameter is needed in the application-class block (line 3). The node ha1 is the active (or primary) node, the node that runs *named* when the nodes are in normal state.

Each application class needs one action block (lines 6 to 9) and one action-timer block (lines 11 to 16). These blocks have standard parameters, which are described in Chapter 4 of the IRIS FailSafe Administrator's Guide.

Because there is just one instance of *named* running in the cluster, there is just one named block (lines 18 to 23). The block's label is named_block (line 18). The block specifies the active node as ha1 (the server-node parameter on line 20) and the backup node as ha1 (the backup-node parameter on line 21). The process-name parameter (line 22) will be used by the local monitoring script /var/ha/actions/ha_named_lmon. (See Example 3-1.)

Adding a New Highly Available Service to the Configuration File

You must develop the configuration file contents for a new highly available service in several phases as you proceed with developing and testing the scripts for the new highly available services. The phases are:

- 1. Put the framework in place in /var/ha/ha.conf. This phase can be done before writing the scripts described in Chapter 3, "Writing a Monitoring Script," and Chapter 4, "Writing a Failover Script." This framework has these components:
 - Begin with a copy of the /var/ha/ha.conf that you are using on your IRIS FailSafe cluster.
 - Add a copy of the template for new highly available applications, the file /var/ha/templates/ha.conf.template.
 - If you know how many instances of the application will be running on the cluster, create that many copies of the <application_name> block. Using filesystems as an example, the <application_name> is "filesystem," and there is one filesystem block for each filesystem on the shared disks in a cluster.
 - Edit the template to fill in as much information as you can. For example, choose a name for your highly available service and substitute it for <application_name> throughout the file. Choose a label for each <application_name> block. Enter the default values for the monitoring frequency parameters in the action-timer block.
- 2. Determine if your new highly available service requires the use of system resources such as IP addresses, filesystems, and XLV logical volumes that are not already in the configuration file. If they are required, add them to the configuration file. Follow the instructions in Chapter 2 and Chapter 4 of the IRIS FailSafe Administrator's Guide.
- 3. Customize the configuration file information for the new highly available application as you develop the monitoring and failover scripts. For example, you can enter the pathname of your local monitoring script. In addition, during script development you may find that you want to add parameters to the configuration file that are specific to your application. The section "Choosing Parameters for a New Highly Available Service" in this chapter describes the type of information specified in parameters, the syntax you must use, and how to name them.
- 4. After you complete the development of the monitoring and failover scripts, but before you install your new configuration file, check the syntax of the new configuration file as described in the section "Checking the Configuration File" in this chapter.

5. As you test the IRIS FailSafe system with the new monitoring and failover scripts, refine the values of the start-monitor-time, lmon-probe-time, and lmon-timeout parameters. Remember that the value of start-monitor-time must be greater than or equal to the value of the long-timeout parameter. The values for lmon-probe-time and lmon-timeout depend on the time it takes to execute the local monitoring script for the application.

Choosing Parameters for a New Highly Available Service

The parameters in the IRIS FailSafe configuration file specify many details for each highly available service, including the number of instances and which node is their primary node, which monitoring script is to be used, and various monitoring frequency values. These parameters are used by monitoring scripts and failover scripts. Placing parameters in the configuration file, rather than simply hardcoding values in the scripts, simplifies maintenance when values are used in more than one script and simplifies tuning by putting all values that are likely to change into just one file.

For each parameter you add, you must choose a name. You have two choices:

- Use a parameter name already in use.
 - Tables in Appendix A, "Names Used in Template Configuration Files," list the names already in use. To simplify maintenance, you should use these names for the purposes listed.
- Choose a new name.

See the section "Defining Variables for New Block, Section, and Parameter Names" in Chapter 3 for more information.

Consider the structure of your blocks when adding parameters. Simple parameter specification is structured like this:

```
block-name {
    parameter-name = value
    . . .
}
```

Parameters can take several values, known as a list:

```
block-name {
    parameter-name = ( value1 value2 ... )
    ...
}

If necessary, you can include sections as follows:
block-name {
    section-name label {
        parameter-name = value
    }
    ...
}
```

The formal syntax definition of the configuration file is given in Chapter 4 of the *IRIS FailSafe Administrator's Guide*.

Checking the Configuration File

You can check the syntax and to some extent the content of the configuration file that you create by entering this *ha_cfgverify* command:

```
# /usr/etc/ha_cfgverify configuration_file
```

configuration_file is the pathname of your configuration file under development. The messages produced by *ha_cfgverify* are described in Appendix A of the *IRIS FailSafe Administrator's Guide*.

Writing a Monitoring Script

This chapter provides information about writing local and remote monitoring scripts. It begins with a section that describes how to write a monitoring script. The remaining sections provide details about various aspects of monitoring scripts that will help you develop your script.

The sections in this chapter are as follows:

- "Writing a Monitoring Script" on page 37
- "Preparing to Write a Monitoring Script" on page 39
- "Understanding the Monitoring Script Template" on page 41
- "Defining Variables for New Block, Section, and Parameter Names" on page 45
- "Using ha_cfginfo to Get Configuration File Information" on page 46
- "Understanding the Function of the Monitoring Script check() Function" on page 48
- "Executing a Command in a Monitoring Script" on page 50

Writing a Monitoring Script

Two types of monitoring scripts can be written. A *local* monitoring script monitors a particular resource, for example network interfaces, or the instances of an application, for example Netscape servers, on the local node. A *remote* monitoring script monitors a resource or the instances of an application on the other node in the cluster. The procedure below describes the steps to write a local or remote monitoring script, including choosing which type of script to write.

- 1. Use the questions and information in the section "Preparing to Write a Monitoring Script" in this chapter to help you get information you may need about the application you are monitoring and make decisions about how to implement monitoring.
- 2. Begin with the monitoring script template /var/ha/actions/ha_app_Xmon or with a copy of one of the monitoring scripts provided with the product if it is similar to the script you need.
- 3. Review the monitoring script template /var/ha/actions/ha_app_Xmon. It is described in the section "Understanding the Monitoring Script Template" in this chapter.
- 4. Become familiar with using the command *ha_cfginfo* to extract information from the configuration file. The use of this command is described in the section "Using ha_cfginfo to Get Configuration File Information" in this chapter.
- 5. Review other monitoring scripts in /var/ha/actions to get an idea of how they perform their checking tests and to see if portions of these scripts can be reused in your script.
- 6. Define variables in the monitoring script for any new block, section, and parameter names you added to the configuration file that you will use in the monitoring script. See the section "Defining Variables for New Block, Section, and Parameter Names" in this chapter for details.
- 7. Write the function that performs the failure test, **check()**. It should contain code that searches for all instances of the application running on \$HOST and, for each instance of the application, perform the failure test. The requirements for this function are described in the section "Understanding the Function of the Monitoring Script check() Function" in this chapter.
- 8. Modify the remainder of the script as necessary.

The remaining sections in this chapter provide information that will help you perform these steps.

Preparing to Write a Monitoring Script

Here are some questions to think about before writing a monitoring script:

• Is a monitoring script required?

Monitoring scripts may not be needed at all in these situations:

- Heartbeat monitoring is sufficient; simply verifying that the node is alive (provided automatically by IRIS FailSafe software) determines the health of the highly available service.
- There is no process or resource that can be monitored. For example, the Silicon Graphics Gauntlet software performs IP filtering on firewall nodes. Because the filtering is done in the kernel, there is no process or resource to monitor.
- The resource on which the application depends is already monitored. For example, monitoring some client-server applications might best be done by monitoring the filesystems, volumes, and network interfaces they use. Because this is already done by the IRIS FailSafe base software (the /var/ha/actions/ha_filesys_lmon and /var/ha/actions/ha_vol_lmon scripts and the ha_ifa interface agent), additional monitoring is not required.
- Can a local monitoring script be written?

Local monitoring may be so expensive that it affects system performance. In this case it shouldn't be done. Also, security issues may make monitoring very difficult.

In some unusual situations, applications may not allow local monitoring. For example, the application may prevent local clients from connecting. In this case, only remote monitoring can be done.

Is a remote monitoring script necessary?

There are generally two components to remote monitoring: testing the network between the two nodes and verifying that the application is running in the remote node. Because the network interfaces specified in the node blocks of the configuration file are monitored by the interface agent and the application can be monitored by a local monitoring script, a remote monitoring script may not be necessary.

• What are the symptoms of failure for this application?

Some possibilities include:

- The application returns an error code.
- The application returns the wrong result.

- The application does not return quickly enough.
- What is the test for failure?

The test should be simple and complete quickly, whether it succeeds or fails. Some examples of tests are as follows:

- For a client-server application that follows a protocol, the monitoring script can make a simple request and verify that the proper response is received.
- For a web server, the monitoring script can request a home page, verify that the connection was made, and ignore the resulting home page.
- For a database, a simple request such as querying a table can be made.
- For NFS, more complicated end-to-end monitoring is required. The test might consist of mounting an exported filesystem, checking access to the filesystem with a stat() system call to the root of the filesystem, and undoing the mount.
- For an application that writes to a log file, check that the size of the log file is increasing or use the *grep* command to check for a particular message.
- The command
 - # killall -0 processname

can be used to determine quickly whether a process exists. Using the *ps* command to check on a particular process is not a good test; its execution can be too slow.

• What should the probe time be set to (the frequency of monitoring)?

For local monitoring, the probe time should be a balance between the frequency of checking and the cost of checking. Monitoring reduces the performance of a node.

For remote monitoring, the probe time should be longer than the probe time for local monitoring and longer than the heartbeat probe time. A good initial value for the probe time for remote monitoring is the value of long-timeout. Remote monitoring is much more likely to suffer from timeouts than local monitoring.

• What should the timeout be (the period in which a test should complete)?

This value must be determined by testing the monitoring script. It must be long enough to guarantee that occasional anomalies do not cause false failovers.

• Should the failure test be executed multiple times so that a node is not declared dead after a single failure?

Testing more than once before declaring failure is a good idea. One way to do this if the test is a single command is to use the ha_exec command. It is described in the section "Understanding the Function of the Monitoring Script check() Function."

 What values need to be customized or tuned and should therefore go into the configuration file as parameters?

See Chapter 2, "Modifying the Configuration File for a New Highly Available Service," for information on adding parameters to /var/ha/ha.conf.

Understanding the Monitoring Script Template

The monitoring script template /var/ha/actions/ha_app_Xmon is shown in Example 3-1. A description of the template is provided at the end of the template.

Example 3-1 Monitoring Script Template With Line Numbers

```
1 #!/sbin/sh
 2 #
 3 ## Instructions for modifying this file are on lines that begin with ##.
 4 #
 5 ## Provide a description of this script including its name, installation
 6 ## location, purpose, and the monitoring tests performed.
 7 #
 8 # Usage:
 9 ## Replace <scriptname> in the next line with the name of this script.
10 #
              <scriptname> "<checksum> <nodename>"
11 #
12 # Exit codes:
             0: The local/remote monitor succeeded
13 #
14 #
              1: This script called illegally
15 #
              2: Configuration file is incorrect
16 #
              3: The local/remote monitoring failed
17 #
18
19 SUCCESS=0
20 ILLEGAL_CALL=1
21 INCORRECT_CONF_FILE=2
22 FAILED=3
23
24 HA_DIR=/var/ha/actions
```

```
25 HAEXEC=/usr/etc/ha_exec
 26 CONF=$HA_DIR/common.vars
 27
 28 ## Define other variables that are local to this script here.
 29 ## Use ${LOGGER} to print error and TESTING messages to /var/adm/SYSLOG
 30 ## file.
 31
 32 # Source in common variables
33 . $CONF
 34
 35 if [ X$TESTING = Xok ]; then
 36 ## Replace <application> and <local/remote> in the next line.
        ${LOGGER} "Executing <application> <local/remote> monitor
script"
 38 fi
 39
 40 if [ $# -ne 1 ]; then
        ${LOGGER} "Illegal syntax: argument required"
        ${LOGGER} "Usage: $0 \"checksum nodename\""
 42
 43
        exit $ILLEGAL_CALL;
 44 fi
 45
 46 # Get the checksum and nodename from the argument string.
 47 set $1
 48
 49 if [ $# -ne 2 ]; then
        \{LOGGER\} "Illegal syntax: argument required"
 50
        ${LOGGER} "Usage: $0 \"checksum nodename\""
 51
 52
        exit $ILLEGAL_CALL;
 53 fi
 54
55 HOST=$2
 56
 57 #
 58 # Compare the checksum argument (the checksum known by the node
 59 # controller and application monitor) with the checksum of ha.conf
 60 # on this system.
 61
 62 CNF_CHKSUM=$1
 63 CHKSUM=`$CFG_SUM`
 64 if [ $CNF_CHKSUM != $CHKSUM ]; then
        ${LOGGER} "Checksum mismatch [argument: $CNF_CHKSUM] [file:
 65
$CHKSUM1"
 66
        exit $INCORRECT_CONF_FILE;
 67 fi
```

```
68
 69 ##
 70 ## Substitute ha_app_Xmon by the application name
 71 ##
72 LOGFILE=/var/ha/logs/ha_app_Xmon.$HOST.log
73 echo Started logging at `date` > $LOGFILE
75 #
 76 \# Executes the command \$EXEC and prints the command, output and
 77 # error to log file $LOGFILE. If the return value from the command
 78 # is non-zero, the function exits with value 3.
 79 # It takes one parameter, log message about the command.
 80 #
 81 execute_cmd()
 82 {
83
 84
         echo $1 >> $LOGFILE;
 85
         if [ X${TESTING} = Xok ]; then
 86
             ${LOGGER} $1
 87
         fi
 88
         eval $EXEC >> $LOGFILE 2>&1;
 89
 90
 91
         exit_code=$?;
 92
 93
         if [ $exit_code -ne 0 ]; then
 94
             echo "ERROR: $EXEC, exit_code: $exit_code" >> $LOGFILE;
             ${LOGGER} "ERROR: $EXEC"
 95
 96
             exit 3;
 97
         fi
 98
99
         echo "*** $EXEC completed with exit_code 0 ***" >> $LOGFILE;
100
101 }
102
103 ## Put the checking procedure(s) here.
104
105 ## Comment about check() procedure.
106 ## Use $HAEXEC for commands which have to be retried before declaring
107 ## application monitor failure.
108 ## Check to see if the application instances whose server-node is $HOST
109 ## has failed.
110 ## The check() procedure should return $FAILED if the application
111 ## instance has failed.
112 ## If the configuration file ha.conf is incorrect, check() procedure
```

```
113 ## should return $INCORRECT_CONF_FILE.
114 ## To read the configuration file ha.conf, use $CFG_INFO command. For
115 ## more information about the command, see ha_cfginfo(1M) manpage.
116 ## Use execute_cmd() to execute the commands in the script.
117 ##
118
119 ## check()
120 ## {
121 ##
122 ## }
123
124 ## Make call(s) to checking procedure(s) here.
125
126 ## check;
127
128 # Exit with SUCCESS
129
130 exit $SUCCESS;
```

The monitoring script template can be broken into these sections:

- Lines 19 to 22 set variables for the script return values. Failover scripts have these return values:
 - 0 (\$SUCCESS)—Success; the operation succeeded, so the application is running.
 - 1 (\$ILLEGAL_CALL)—An invalid argument was passed to the script.
 - 2 (\$INCORRECT_CONF_FILE)—The configuration file is invalid; either information in the configuration file is incorrect or some information is missing from the configuration file.
 - 3 (\$FAILED)—The operation failed.

If a monitoring script returns a non-zero value, the application is assumed to have failed.

• Line 33 sources the file /var/ha/actions/common.vars, which assigns strings in /var/ha/ha.conf to variables and defines the \${LOGGER} command, which is used to write messages to /var/adm/SYLOG, and the \${TESTING} variable, which is used to control debugging information written to /var/adm/SYSLOG. It also sets the variable \${CFG_SEP} to the character #.

- Lines 40 to 54 contain code for checking the monitoring script's command-line argument. The monitoring script must have one command line argument, a double-quoted argument that contains two strings separated by a blank:
 - The first string is the checksum of the /var/ha/ha.conf file, as generated by the ha_cfgchksum command.
 - The second string is a node name. This is the hostname of the node to be monitored. Line 55 sets \$HOST to the node name.
- Lines 62 to 67 compare the checksum argument with the checksum of /var/ha/ha.conf.
- Lines 72 and 73 set \$LOGFILE to the name of the log file and write a message to it. The directory for the log file is /var/ha/logs. The convention for the filename is the name of the application, \$HOST, and the word log, separated by periods.
- Lines 76 to 101 describe and define the execute_cmd() function. It writes
 information to the log file and executes the command specified by the variable
 \$EXEC. It is described fully in the section "Executing a Command in a Monitoring
 Script."
- Lines 103 to 122 describe and define the **check()** function. The **check()** function is described fully in the next section, "Understanding the Function of the Monitoring Script check() Function."

Defining Variables for New Block, Section, and Parameter Names

Each new block, section, or parameter name that you added to the configuration file (see the section "Choosing Parameters for a New Highly Available Service" in Chapter 2) must be assigned to a shell variable at the beginning of each script in which they are used. The variables are used in scripts, not the parameter, section, and block names.

For example, say that you added this parameter to the configuration file:

```
process-name = named
```

To use this parameter in a script, add this line to the script you write (about line 31 in Example 3-1):

```
T_PROCNAME=process-name
```

Using ha_cfginfo to Get Configuration File Information

The command *ha_cfginfo* is used in monitoring and failover scripts to obtain information from the configuration file */var/ha/ha.conf*. The command is

```
# /usr/etc/ha_cfginfo [ -f filename ] [ string ]
```

filename is the name of a configuration file; by default it is /var/ha/ha.conf. If string isn't specified, the names of the blocks in the configuration file are listed. By specifying string, you can get any value in the file. For example, say that this is a portion of a configuration file:

```
volume shared1_vol
{
    server-node = xfs-ha1
    backup-node = xfs-ha2
    devname = /dev/dsk/xlv/shared1_vol
}

volume shared2_vol
{
    server-node = xfs-ha2
    backup-node = xfs-ha1
    devname = /dev/dsk/xlv/shared2_vol
    disks = (/dev/dsk/dks0dls2 /dev/dsk/dks0d5s3 /dev/dsk/dks0d2s6)
}
```

Some example *ha_cfginfo* commands and their output are shown below. The *string* argument specifies the hierarchical path you are interested in, with the # character separating elements in the hierarchy.

```
# /usr/etc/ha_cfginfo volume
shared1_vol shared2_vol
# /usr/etc/ha_cfginfo volume#shared1_vol
server-node backup-node devname
```

```
# /usr/etc/ha_cfginfo volume#shared1_vol#server-node
xfs-ha1
# /usr/etc/ha_cfginfo volume#shared2_vol#server-node
xfs-ha2
# /usr/etc/ha_cfginfo volume#shared2_vol#disks
/dev/dsk/dks0d1s2 /dev/dsk/dks0d5s3 /dev/dsk/dks0d2s6
```

A simple example of using *ha_cfginfo* in a script is this fragment that monitors each of the volumes defined in */var/ha/ha.conf*:

```
EXEC = \'/usr/etc/ha_cfginfo volume\'
for VOL in $EXEC
do
    monitor the volume $VOL here
done
```

Scripts access the labels and parameter values in /var/ha/ha.conf by specifying the hierarchical path to the label or parameter they want—for example, the block, its label, a section, its label, and finally the parameter—as an argument to the ha_cfginfo command. However, there is a level of indirection in the naming of the blocks, sections, and parameters. In the shell script /var/ha/actions/common.vars, each block, section, and string name in /var/ha/ha.conf is assigned to a similarly named variable. These variables are used as arguments to ha_cfginfo in monitoring and failover scripts.

As an example of the use of *ha_cfginfo*, say that the configuration file contains this fragment:

```
nfs nfs1
{
     export-point = /shared1/export
     ...
}
nfs nfs2
{
     export-point = /shared2/export
     ...
}
```

The file /var/ha/actions/common.vars includes these lines:

```
CFG_FILE=/var/ha/ha.conf
CFG_INFO="/usr/etc/ha_cfginfo -f ${CFG_FILE}"
CFG_SEP=#
T_NFS=nfs
T_EXPORTPT=export-point
```

To perform an operation on each export point for NFS filesystems, use a shell script fragment such as this to get the value of each export-point parameter:

```
for FS in `$CFG_INFO ${T_NFS}` # loop through each nfs block
do
# set up the ha_cfginfo command line to get the export-point value of an nfs block
SEARCH="$CFG_INFO ${T_NFS}${CFG_SEP}${FS}${CFG_SEP}$${T_EXPORTPT}"
# perform the ha_cfginfo command, assign the result to $EXPORT_PT
EXPORT_PT=`$SEARCH`
# perform operation on $EXPORT_PT
...
done
```

Understanding the Function of the Monitoring Script check() Function

The checking function **check()** must perform these functions:

- Check to see if the application instances whose server-node is \$HOST have failed.
- Exit the script with the return value \$FAILED if the application instance has failed.
- Exit the script with the return value \$INCORRECT_CONF_FILE if the configuration file /var/ha/ha.conf is incorrect.

To extract information from <code>/var/ha/ha.conf</code>, use the <code>ha_cfginfo</code> command. (The <code>common.vars</code> file sets the variable \$CFG_INFO to the <code>ha_cfginfo</code> command.) <code>ha_cfginfo</code> is described in the section "Using ha_cfginfo to Get Configuration File Information" in this chapter.

When executing each command used to check if an application instance has failed, you can use the *ha_exec* command, which provides automatic retry and timeout, and the **execute_cmd()** function, which provides automatic logging. See the subsection "Executing a Command in a Monitoring Script" for more information.

Shown below is the **check()** function for a *named* local monitoring script (to be installed as /var/ha/actions/ha_named_lmon).

```
check()
    NAMED=named
    # for each named block ...
    for i in `$CFG_INFO ${NAMED}`
    dО
         # get the server-node name
         SEARCH="$CFG_INFO ${NAMED}${CFG_SEP}${i}${CFG_SEP}${T_SERVER}"
         SERVER_NODE=`$SEARCH`
         # if that failed, log a message and exit
         if [ $? -eq 1 ]; then
    ${LOGGER} "$0: Trouble finding server node for named $i ($SEARCH)"
             exit $INCORRECT_CONF_FILE;
         fi
         # if this node is the server-node ...
         if [ X${SERVER_NODE} = X${HOST} ]; then
             # get the value of process-name
         SEARCH="$CFG_INFO ${NAMED}${CFG_SEP}${i}${CFG_SEP}${PROC_NAME}"
             PROC_NAME=`$SEARCH`
             # if that failed, log a message and exit
             if [ $? -eq 1 ]; then
    ${LOGGER} "$0: Trouble finding process name for named $i ($SEARCH)"
                  exit $INCORRECT_CONF_FILE;
             fi
             # set up and execute the command "killall -0 named", which checks to
             # see if named is running
             EXEC="${KILLALL} -0 ${PROC_NAME}"
             execute_cmd "check if ${PROC_NAME} is running"
         fi
    done
```

Executing a Command in a Monitoring Script

To execute each command that you add to a script, you have these choices:

- Execute the command.
- Use the *ha_exec* command to execute the command.

ha_exec is used when the command has to be retried before declaring that the application has failed or when the command might not return quickly enough and you want to set a time limit.

The syntax of the *ha_exec* command is

```
ha_exec [ -p waitperiod ] timeout retry command
```

command is the command for the failure test, timeout is the length of time to wait without response before declaring that a single test failure, retry is the number of times to retry the test, and waitperiod is the length of time to wait after a failure or command timeout before retrying command. waitperiod defaults to 0. (See the ha_exec(1M) reference page for more information.)

• Use the **execute_cmd()** function (with or without *ha_exec*) to execute the command.

The **execute_cmd()** function writes information to the log file and executes the command specified by the variable \$EXEC. It takes one parameter, a string that is a message of your choice. It executes a command you specify and writes a message passed as a parameter, the command executed, the output of the command executed, and a message about the return value of the command to the log file /var/ha/logs/ha_<app>_lmon.<node_name>.log. This log file makes debugging a monitoring script failure easier. The command is the value of \$EXEC, which you set in the **check()** function.

For example, say that you decide to use this command to determine if the *sendmail* process is running: *killall -0 sendmail*. This command returns 0 if *sendmail* is running and non-zero if it is not. Your choices are these:

• Execute the command and check the return value with code such as this:

```
RESULT = 'killall -0 sendmail'
```

• Use *ha_exec* to execute the command, giving it three seconds to return and trying twice if necessary:

```
RESULT = '$HAEXEC 3 2 "killall -0 sendmail"'
```

• Use **execute_cmd()** without *ha_exec* to execute the command:

```
EXEC = "killall -0 sendmail"
RESULT = execute_cmd "checking for sendmail"
```

• Use **execute_cmd()** with *ha_exec* to execute the command:

```
EXEC = '$HAEXEC 3 2 "killall -0 sendmail"'
RESULT = execute_cmd "checking for sendmail"
```

Using **execute_cmd()** with *ha_exec* is recommended.

When choosing between these different methods of executing a command, keep these things in mind:

- Use *ha_exec* when the command might fail and has to be retried or when the command might not return quickly and you want to set a time limit.
- When you use *ha_exec* and **execute_cmd()**, the command must return 0 on success and non-zero on failure.
- If you need to examine the output of the command, don't use **execute_cmd()** because the output goes to the log file, where it would be difficult to parse.

Writing a Failover Script

This chapter provides information about writing a failover script for a resource or application that you want the IRIS FailSafe system to treat as a highly available service. It begins with a section that describes how to write a failover script. The remaining sections provide details about various aspects of failover scripts that will help you develop your script.

The sections in this chapter are as follows:

- "Writing a Failover Script" on page 53
- "Preparing to Write a Failover Script" on page 54
- "Understanding the Failover Script Template" on page 56
- "Writing the Failover Functions" on page 62
- "Executing a Command in a Failover Script" on page 64

Writing a Failover Script

Follow these steps to write a failover script:

- 1. Use the questions and information in the section "Preparing to Write a Failover Script" in this chapter to help you get information about the application you are failing over and make decisions about how to implement the failover script.
- 2. Begin with the failover script template in /var/ha/resources/appclass or with a copy of one of the failover scripts in /var/ha/resources if it is similar to the script you need.
- 3. Review the failover script template /var/ha/resources/appclass. It is described in the section "Understanding the Failover Script Template" in this chapter.
- 4. If necessary, review how to extract information from /var/ha/ha.conf using the ha_cfginfo command. ha_cfginfo is described in the section "Using ha_cfginfo to Get Configuration File Information" in Chapter 3.

- 5. Review your choices for executing commands in the script, which are described in the section "Executing a Command in a Failover Script" in this chapter.
- 6. Define variables in the monitoring script for any new block, section, and parameter names you added to the configuration file that you will use in the monitoring script. See the section "Defining Variables for New Block, Section, and Parameter Names" in Chapter 3 for details.
- 7. Write the **takeback()**, **takeover()**, **giveaway()**, and **giveback()** functions. See the section "Writing the Failover Functions" in this chapter for more information.
- 8. Make each of the functions takeback(), takeover(), giveaway(), and giveback() in the script idempotent—if it is executed twice in a row and the first execution succeeds, the second time must also succeed.
 - For example, running the script with the **giveaway** argument should stop all instances of the highly available service. If it is run again immediately, it should return without error. If the **giveaway** argument is specified when no instances of highly available service are running, the **giveaway**() function must succeed. To achieve this, you may have to add a check that tests whether the application is running prior to each command that halts an application. The command to halt an application is executed only if the application is running.
- 9. Review each function and test if necessary to verify that it executes in less than the value of the long-timeout parameter in the internal block, which is 60 seconds by default.

The remaining sections in this chapter provide information that will help you perform these steps.

Preparing to Write a Failover Script

Each highly available service has a failover script in the /var/ha/resources directory. This script contains at least these four functions: takeover(), takeback(), giveaway(), and giveback().

Here are some questions to think about before writing a failover script:

- How do I move this application from one machine to another?
- Can this application be moved at any time?

- Do any highly available services, such as filesystems on shared disks, need to exist before the application can be started on another node?
- Do any actions need to be performed to recover lost transactions, data, or state before starting the application on another node?
 - For example, databases are able to recover lost transactions. Commands can be executed by the script to recover lost transactions. For NFS filesystems or a Netscape server, there is no automatic recovery; the client simply requests the data again.
- How do you start and stop the application on a node? How do you start and stop a specific instance of the application?
- Can the application be started and stopped as root or must it be another user?
- If a user other than root must start and stop the application, should that user be specified in /var/ha/ha.conf?
- Where is the configuration information for the application stored? Will it be on shared or local disks?
 - You may not have any flexibility about where the configuration information is stored. To store it on a shared disk, you may need to link or copy files. (Remember that shared disks don't allow concurrent access; they can be used by only one node at a time.)
- Where is the data for the application stored? Will it be on shared or local disks?
- Does the application both read and write data or just read it?
 If the application doesn't write data, for example a front end Web server that has ready-only data, duplicating the data on local disks might be the best choice.
- Where is the log information for the application stored? Will it be on shared or local disks?
- Where is the application itself stored? Will it be on shared or local disks?
- Does information about the application, data, log, or configuration information need to be specified in /var/ha/ha.conf?

- What tasks will be performed by other failover scripts executed prior to this script?
 - If log, data, or configuration information is stored in a raw volume or in a filesystem on a shared disk, the *filesystems* and *volumes* failover scripts must be run before the application is started by the takeover or takeback operations.
 - See "Tasks Performed by the Standard Failover Scripts" in Chapter 1 for information about the actions of each failover script.
- What tasks will be performed by other failover scripts after this script is executed?
 - If log, data, or configuration information is stored in a raw volume or in a filesystem on a shared disk, the *filesystems* and *volumes* failover scripts must be run after the application is stopped by the giveback or giveaway operations.
 - See "Tasks Performed by the Standard Failover Scripts" in Chapter 1 for information about the actions of each failover script. See "Choosing the Execution Order of Failover Scripts for Each Operation" in Chapter 5 for information about specifying the ordering of execution of the script relative to other scripts.
- What additional information about the application should be stored in /var/ha/ha.conf?

All shared filesystems and volumes must be specified in /var/ha/ha.conf. Command-line arguments for starting and stopping applications should be put in /var/ha/ha.conf if they will vary; otherwise they can be hardcoded in the failover script.

Understanding the Failover Script Template

The failover script template /var/ha/resources/appclass is shown in Example 4-1. A description of the template is provided at the end of the template.

Example 4-1 Failover Script Template

- 1 #!/sbin/sh
- 2 #
- 3 ## Instructions for modifying this file are on lines that begin with ##.
- 5 ## Provide a description of this script including its name, installation
- 6 ## location, purpose and the resource(s)/application(s) that it fails over.
- 7 #
- 8 # Usage:
- 9 ## Replace <scriptname> in the next line with the name of this script.

```
10 #
            <scriptname> <checksum> <argument>
11 #
12 #
       The <argument> can be one of the operations - giveback,
13 #
       giveaway, takeover or takeback.
 14 #
 15 # Exit codes:
          0: The operation succeeded
 16 #
 17 #
          1: This script called illegally
          2: Configuration file is incorrect
 18 #
          3: Command exited with non-zero return code - the action
19 #
 20 #
             failed.
 21
 22 SUCCESS=0
 23 ILLEGAL_CALL=1
 24 INCORRECT_CONF_FILE=2
 25 FAILED=3
 27 HA DIR=/var/ha/actions
 28 CONF=$HA_DIR/common.vars
 29
 30 ## Define other variables that are local to this script here.
 31 ## Use ${LOGGER} to print error and TESTING messages to /var/adm/SYSLOG
 32 ## file.
33
 34 # Source in common variables
 35 . $CONF
 36
 37 if [ X$TESTING = X"ok" ]; then
 38 ## Replace <application> in the next line.
        ${LOGGER} "Executing <application> script"
 40 fi
 41
 42 if [ $# -ne 2 ]; then
        ${LOGGER} "Illegal syntax: checksum and argument required"
 43
        ${LOGGER} "Usage: $0 <checksum> <argument>"
 45
        exit $ILLEGAL_CALL;
 46 fi
 47
 48 if [ $2 != "giveback" -a $2 != "takeback" -a $2 != "takeover" -a
$2 != "giveaway" ]; then
        ${LOGGER} "Illegal argument: must be giveback, giveaway,
takeback, or takeover"
        ${LOGGER} "Usage: $0 <checksum> <argument>"
 51
        exit $ILLEGAL_CALL;
 52 fi
```

```
53
 54 #
 55 # Compare the checksum argument (the checksum known by the node
 56 # controller and application monitor) with the checksum of ha.conf
 57 # on this system.
 58 CNF CHKSUM=$1
 59 CHKSUM=`$CFG_SUM`
 60 if [ $CNF_CHKSUM != $CHKSUM ]; then
        \{LOGGER\}\ "Checksum mismatch [argument: $CNF_CHKSUM]\ [file:
$CHKSUM]"
 62
        exit $INCORRECT_CONF_FILE;
 63 fi
 64
 65 HOST=`hostname`
 66
 67 ##
 68 ## Substitute applclass by the application name
 69 ##
70 LOGFILE=/var/ha/logs/appclass.log
71 echo Started logging at `date` > $LOGFILE
72
73 #
74 # Executes the command $EXEC and prints the command, output and
 75 # error to log file $LOGFILE. If the return value from the command
76 # is non-zero, the function exits with value 3.
77 # It takes one parameter, log message about the command.
 78 #
 79 execute_cmd()
 80 {
 81
 82
        echo $1 >> $LOGFILE;
 83
        if [ X${TESTING} = Xok ]; then
 84
            ${LOGGER} $1
 85
        fi
 86
 87
        eval $EXEC >> $LOGFILE 2>&1;
 88
 89
        exit_code=$?;
 90
 91
        if [ $exit_code -ne 0 ]; then
 92
            echo "ERROR: $EXEC, exit_code: $exit_code" >> $LOGFILE;
 93
            ${LOGGER} "ERROR: $EXEC"
 94
            exit 3;
 95
        fi
 96
```

```
97
          echo "*** $EXEC completed with exit_code 0 ***" >> $LOGFILE;
 98
 99
100
101 ## Put the procedures here.
102
103 ## Comment about giveback() procedure.
104 ## Stop all the application instance(s) or resource(s) for which $HOST
105 ## is the backup-node.
106 ## Use $CFG_INFO to read information from the configuration file, ha.conf.
107 ## To get more information about the command, see ha_cfginfo(1M) manpage.
108 ## The procedure should return $FAILED on failure and $SUCCESS on
109 ## success of the operation.
110 ## giveback()
111 ## {
112 ##
113 ## }
114
115 ## Comment about giveaway() procedure.
116 ## Stop all the application instance(s) or resource(s) for which $HOST
117 ## is the server-node.
118 ## Use $CFG_INFO to read information from the configuration file, ha.conf.
119 ## The procedure should return $FAILED on failure and $SUCCESS on
120 ## success of the operation.
121 ## giveaway()
122 ## {
123 ##
124 ## }
125
126 ## Comment about takeover() procedure.
127 ## Start all the application instance(s) or resource(s) for which $HOST
128 ## is the backup-node.
129 ## Use $CFG_INFO to read information from the configuration file, ha.conf.
130 ## The procedure should return $FAILED on failure and $SUCCESS on
131 ## success of the operation.
132 ## takeover()
133 ## {
134 ##
135 ## }
136
137 ## Comment about takeback() procedure.
138 ## Start all the application instance(s) or resource(s) for which $HOST
139 ## is the server-node.
140 ## Use $CFG_INFO to read information from the configuration file, ha.conf.
141 ## The procedure should return $FAILED on failure and $SUCCESS on
```

```
142 ## success of the operation.
143 ## takeback()
144 ## {
145 ##
146 ## }
147
148 ## Make calls to operation procedures here.
149
150 if [ $2 = "giveback" ]; then
151
       giveback;
152 elif [ $2 = "takeover" ]; then
        takeover;
154 elif [ $2 = "takeback" ]; then
155
       takeback;
156 elif [ $2 = "giveaway" ]; then
157
        qiveaway;
158 fi
159
160 # Exit with SUCCESS.
161
162 exit $SUCCESS;
```

The failover script template can be broken into these sections:

- Lines 22 to 25 set variables for the script return values. Failover scripts have these return values:
 - 0 (\$SUCCESS)—Success; the operation succeeded.
 - 1 (\$ILLEGAL_CALL)—An invalid argument was passed to the script.
 - 2 (\$INCORRECT_CONF_FILE)—The configuration file is invalid; either information in the configuration file is incorrect, some information is missing from the configuration file, or the configuration file changed between starting up IRIS FailSafe and the execution of the script.
 - 3 (\$FAILED)—The operation failed.

If the failover script returns a non-zero value, the script is assumed to have failed.

- Line 35 sources the file /var/ha/actions/common.vars, which assigns strings in /var/ha/ha.conf to variables and defines the \${LOGGER} command, which is used to write messages to /var/adm/SYLOG, and the \${TESTING} variable, which is used to control debugging information written to /var/adm/SYSLOG. It also sets the variable \${CFG_SEP} to the character #.
- Lines 37 to 40 write a message to /var/adm/SYLOG.

- Lines 42 to 63 check the command-line arguments. The script takes two arguments:
 - The first argument is the checksum for the configuration file. Lines 58 to 63 compare the checksum argument with the checksum of /var/ha/ha.conf.
 - The second argument is an operation: **takeback**, **takeover**, **giveaway**, or **giveback**. Lines 48 to 52 check this argument.
- Line 65 sets \$HOST to the result of the *hostname* command.
- Lines 70 and 71 set \$LOGFILE to the name of the log file and write a message to it. The directory for the log file is /var/ha/logs. The convention for the filename is the name of the application and the word log, separated by a period.
- Lines 76 to 101 describe and define the execute_cmd() procedure. It takes one parameter, a string that is a message of your choice. It executes a command you specify and writes a message passed as a parameter, the command executed, the output of the command executed, and a message about the return value of the command to a log file. The command is the value of \$EXEC, which you set in the check() procedure.
- Lines 103 to 113 describe and define the giveback() procedure. The giveback() procedure is described fully in the next subsection, "Writing the Failover Functions."
- Lines 115 to 124 describe and define the giveaway() procedure. The giveaway() procedure is described fully in the next subsection, "Writing the Failover Functions."
- Lines 126 to 135 describe and define the takeover() procedure. The takeover() procedure is described fully in the next subsection, "Writing the Failover Functions."
- Lines 137 to 146 describe and define the takeback() procedure. The takeback() procedure is described fully in the next subsection, "Writing the Failover Functions."
- Lines 150 to 158 call one of the failover functions—the one passed as an argument to the failover script.
- Line 162 exits with the success return value.

Writing the Failover Functions

This section describes how to write the takeback(), takeover(), giveaway(), and giveback() functions. The purposes of these functions are described below:

takeback()	Starts all instances of the applications class for which this node is the primary node.
takeover()	Starts all instances of the applications class for which this node is the backup node.
giveaway()	Stops all instances of the applications class for which this node is the primary node.
giveback()	Stops all instances of the applications class for which this node is the backup node.

As an example, this section uses the *named* daemon as the application to be failed over. It will be run in an active/backup configuration—only one instance of *named* runs on the cluster. Follow these general steps to write the failover functions:

1. Determine the commands required to start and stop instances of the application.

Looking in /etc/init.d/network, which normally starts named in a standalone system, the command to start named is

```
/usr/sbin/named `cat /etc/config/named.options 2> /dev/null` < \
/dev/null</pre>
```

The command to stop *named* is

```
/sbin/killall -k 1 -TERM named
```

2. Review the application's use of configuration files and their locations (on shared disks or non-shared disks?).

For example, *named* uses the configuration files /etc/named.boot and /etc/config/named.options. These files reside on non-shared disks and are identical on each node. Thus, *named* is not dependent upon filesystems that must be failed over.

3. Develop the **takeback()** function. Shown below is the body of this function for *named*, along with line numbers and comments.

```
1
        NAMED=named
 2
        # for each named block ...
 3
        for i in `$CFG_INFO ${NAMED}`
 4
 5
            # get the server-node name
 6 SEARCH="$CFG_INFO ${NAMED}${CFG_SEP}${i}${CFG_SEP}${T_SERVER}"
            SERVER_NODE=`$SEARCH`
 8
            # if that failed, log a message and exit
9
            if [ $? -eq 1 ]; then
10 ${LOGGER} "$0: Trouble finding server node for named ($SEARCH)"
11
                 exit $INCORRECT_CONF_FILE;
12
            fi
13
14
            # if server-node matches $HOST ...
15
            if [ X${SERVER\_NODE} = X$HOST ]; then
16
17
                 # execute the command that starts the application
18
                 EXEC="/usr/sbin/named `cat /etc/config/named.options
2> /dev/null` < /dev/null"</pre>
19
                 execute_cmd "${EXEC}"
20
            fi
21
        done
22
        # exit with success
23
        exit $SUCCESS;
24 }
```

4. Develop the **takeover()** function. It is the same as the **takeback()**, with this exception:

```
# get the backup-node name
SEARCH="$CFG_INFO ${NAMED}${CFG_SEP}${i}${CFG_SEP}${T_BACKUP}"
```

5. Develop the **giveaway()** function. It is the same as the **takeback()**, with this exception:

```
# execute the command that stops the application

EXEC="/sbin/killall -k1 -TERM named";
```

6. Develop the **giveback()** function. It is the same as the **takeback()**, with these exceptions:

```
# get the backup-node name
SEARCH="$CFG_INFO ${NAMED}${CFG_SEP}${i}${CFG_SEP}${T_BACKUP}"

# execute the command that stops the application
EXEC="/sbin/killall -k1 -TERM named";
```

Executing a Command in a Failover Script

To execute each command you add to a script, you have these choices:

- Execute the command.
- Use the **execute_cmd()** function to execute the command.

The execute_cmd() function writes information to the log file and executes the command specified by the variable \$EXEC. It takes one parameter, a string that is a message of your choice. It executes a command you specify and writes a message passed as a parameter, the command executed, the output of the command executed, and a message about the return value of the command to a log file in /var/ha/logs/<application_class>.log. The command is the value of \$EXEC, which you set in the takeback(), takeover(), giveaway(), or giveback() function.

Installing and Testing Scripts

This chapter describes how to name and install new scripts and how to test them. It also provides tips on how to debug problems that you may encounter.

The major sections in this chapter are as follows:

- "Naming and Installing Monitoring Scripts" on page 65
- "Choosing the Execution Order of Failover Scripts for Each Operation" on page 66
- "Installing Failover Scripts" on page 67
- "Modifying Application Startup Procedures" on page 67
- "Testing New Scripts" on page 68

Naming and Installing Monitoring Scripts

Install monitoring scripts in the directory /var/ha/actions with owner root, group sys, and mode 700. Local monitoring scripts have a name of this form:

ha_service_lmon

Remote monitoring scripts have a name of this form:

ha_service_rmon

Choosing the Execution Order of Failover Scripts for Each Operation

The section "IRIS FailSafe Scripts" in Chapter 1 described the organization of the failover scripts:

- The scripts are stored in /var/ha/resources.
- Each of the directories /var/ha/actions.d/giveaway, /var/ha/actions.d/giveback, /var/ha/actions.d/takeback, and /var/ha/actions.d/takeover contains links to each of the scripts.
- The link names begin with S and a three-digit number.
- Because the links (scripts) in each directory are executed in lexicographic order, the
 ordering of the three-digit numbers is the order in which the scripts are executed.

In the *giveaway* and *giveback* directories, the order of execution of the standard, NFS, and Web server scripts is *webserver*, *interfaces*, *statd*, *nfs*, *filesystems*, and *volumes*. In the *takeback* and *takeover* directories, the order of execution is *volumes*, *filesystems*, *nfs*, *interfaces*, *statd*, and *webserver*.

Based on the tasks performed by each of these scripts (see the section "Tasks Performed by the Standard Failover Scripts" in Chapter 1) and the resources used by the application, you must choose where in the sequence of execution to insert your new script for each operation. For example, filesystems on which an application depends must be mounted before the application is started up. Thus, for takeover and takeback operations, the sequence number of the *filesystem* script

(/var/ha/actions.d/takeover/S100filesystem) must be smaller than that of your new highly available service, so that filesystems are mounted before instances of the new highly available service are started up. Similarly, the application sequence number must be smaller than the *filesystem* sequence number for giveback and giveaway operations because the application must be stopped before filesystems are unmounted.

For a failover script for *named*, good choices are as follows:

/var/ha/actions.d/takeback/S850named /var/ha/actions.d/takeover/S850named /var/ha/actions.d/giveback/S700named /var/ha/actions.d/giveaway/S700named

This ordering was chosen because the named process has to be started after the interfaces have been brought up and before NFS filesystems are mounted. It has to be stopped before interfaces are stopped and after NFS filesystems are unmounted.

For most applications it is best not to insert them into the middle of this application order; they should be executed before the scripts provided by Silicon Graphics in the *giveaway* and *giveback* directories and after the scripts provided by Silicon Graphics in the *takeback* and *takeover* directories. Thus, for *giveaway* and *giveback*, applications are stopped before stopping interfaces, filesystems, and volumes. For *takeback* and *takeover*, applications are started after the volumes, filesystems, and interfaces are started.

Installing Failover Scripts

After deciding the execution order of your failover script in each of the *actions.d* directories as described in the section "Choosing the Execution Order of Failover Scripts for Each Operation" in this chapter, you can complete the installation of your script:

- 1. Copy the script to /var/ha/resources.
- 2. Change the owner and group of the script to **root sys** and the mode to 700.
- 3. Choose a three-digit number that will ensure that the script is executed in the correct order in the *giveaway* and *giveback* directories.
- 4. Create links in the *giveaway* and *giveback* directories. In each of these directories, enter this command:
 - # ln -s ../resources/script Snnnscript
- 5. Choose a three-digit number that will ensure that the script is executed in the correct order in the *takeback* and *takeover* directories.
- 6. Create links in the *takeback* and *takeover* directories. In each of these directories, enter this command:
 - # ln -s ../resources/script Smmmscript

Modifying Application Startup Procedures

Because highly available services are started up by IRIS FailSafe, rather than as a result of executing scripts in /etc/init.d or other automatic, non IRIS FailSafe mechanism, you must disable the normal startup procedure for the application you are making highly available.

For example, to turn off the automatic (non IRIS FailSafe) startup of *named*, use the *chkconfig* command to turn *named* **off**:

chkconfig named off

Testing New Scripts

The subsections below describe strategies for testing new monitoring and failover scripts. To prepare for testing, take these steps:

- Ensure that you have exclusive use of both nodes—users logged in during testing could experience unavailability of highly available services.
- Generate additional debugging information in /var/adm/SYSLOG by setting the variable TESTING in /var/ha/actions/common.vars:

TESTING=ok

General Testing and Debugging Techniques

Some general testing and debugging techniques you can use during testing are as follows:

- While testing your scripts, you can get debugging information from these sources:
 - IRIS FailSafe writes messages in /var/adm/SYSLOG, which can be useful in debugging script problems. Running this command in a window dedicated to this command can help you keep track of the messages as they occur:
 - # tail -f /var/adm/SYLOG
 - The *ha_admin -i* command reports the state of a node. Note that this command hangs if a node is in transition from one state to another.
 - The ha_admin -a command provides information about the cluster that includes node states for each node, IP addresses and the node that owns them, XLV volumes and the node that owns them, and filesystems and the node that owns them.

- If your testing causes repeated failovers, IRIS FailSafe is disabled (*chkconfig failsafe off*), so that it is not started automatically at boot time. This is because IRIS FailSafe software is designed so that repeated failures don't result in repeated failovers. The criterion for disabling IRIS FailSafe is two failures within a set period of time. This period of time is specified by the variable MIN_UPTIME in the file /etc/init.d/failsafe. During testing, you can set MIN_UPTIME = 0, with the result that IRIS FailSafe is never disabled.
- The procedures in the following subsections assume that you are using *csh*. If you are using *sh*, substitute *echo* \$? for the *echo* \$status commands that report the return value of the previous command. The return value should always be zero, which indicates success.
- To check that an application is running on a node, you may be able to use a command provided by the application. For example, the IRIS FailSafe INFORMIX option uses the INFORMIX command *onstat*.
- Another way to check that an application is running on a node, is to enter this command on that node:

```
# ps -ef | grep application
```

application is the name (or a portion of the name) of the executable for the application.

Testing Monitoring Scripts

Monitoring scripts test the liveliness of applications and resources. The best way to test them is to induce failures, one at a time, run the script, and check if this failure is detected by the script. Test monitoring scripts without IRIS FailSafe running on either node.

Use this checklist for testing a monitoring script:

- Verify that the script detects failure of the application successfully.
- Verify that the script always exits with a return value. See the section "Understanding the Monitoring Script Template" in Chapter 3 for a list of return values.
- Verify that the script does not contain commands that can hang, such as using DNS for name resolution, or those that continue forever, such as *ping*.
- Verify that the script completes before the timeout value specified in the configuration file.

Verify that the script's return codes are correct.

During testing, measure the time it takes for a script to complete and adjust the monitoring times in the configuration file, /var/ha/ha.conf, accordingly. To get a good estimate of the time required for the script to execute, run it under different system load conditions.

Testing Failover Scripts Without Starting IRIS FailSafe

You can test the operations giveaway, giveback, takeback, and takeover manually using the general procedure below. It refers to one node (either one) as Node A and the other as Node B.

- 1. Before beginning this testing, ensure that the following are true:
 - The failover script you are testing is installed.
 - The configuration file (/var/ha/ha.conf) includes blocks for the application whose script you are testing.
 - IRIS FailSafe is not running on the cluster.
 - The application you are testing starts and stops correctly on each node.
 - The application you are testing is not running on either node in the cluster.
 - The logical volumes used by the application are assembled.
 - The filesystems used by the application are mounted.
 - The network interfaces used the application are configured up.
- 2. On each node, enter this command and check the return value:
 - # /var/ha/actions/takeback \'usr/etc/ha_cfgchksum\'
 - # echo \$status

See the section "General Testing and Debugging Techniques" for information about the *echo* command.

- 3. On each node, verify that all instances of the application for which this node is the primary node (server-node) have been started. See the section "General Testing and Debugging Techniques" for information.
- 4. On Node A, enter this command and check the return value:
 - # /var/ha/actions/giveaway \'usr/etc/ha_cfgchksum\'
 - # echo \$status

- 5. Verify that no instances of the application are running on node A.
- 6. On Node B, enter this command and check the return value:
 - # /var/ha/actions/takeover '/usr/etc/ha_cfgchksum'
 - # echo \$status
- 7. Verify that all instances of the application for which node B is the backup node are now running on Node B.
- 8. On Node B, enter this command and check the return value:
 - # /var/ha/actions/giveback \'usr/etc/ha_cfgchksum\'
 - # echo \$status
- 9. Verify that Node B is running just the application instances for which it is the primary node.
- 10. On Node A, enter this command and check the return value:
 - # /var/ha/actions/takeback \'usr/etc/ha_cfgchksum\'
 - # echo \$status
- 11. Verify that Node A is running the application instances for which it is the primary node.
- 12. On Node B, enter this command and check the return value:
 - # /var/ha/actions/giveaway \'usr/etc/ha_cfgchksum\'
 - # echo \$status
- 13. Verify that no instances of the application are running on node B.
- 14. On Node A, enter this command and check the return value:
 - # /var/ha/actions/takeover '/usr/etc/ha_cfgchksum'
 - # echo \$status
- 15. Verify that all instances of the application for which node A is the primary or backup node are now running on Node A.
- 16. On Node A, enter this command and check the return value:
 - # /var/ha/actions/giveback \'usr/etc/ha_cfgchksum\'
 - # echo \$status
- 17. Verify that Node A is running just the application instances for which it is the primary node.
- 18. On Node B, enter this command and check the return value:
 - # /var/ha/actions/takeback \'usr/etc/ha_cfgchksum\'
 - # echo \$status

19. Verify that Node B is running the application instances for which it is the primary node.

Testing Failover Scripts While Running IRIS FailSafe

You can test the operations giveaway, giveback, takeback, and takeover while IRIS FailSafe is running using the general procedure below. It refers to one node (either node) as Node A and the other as Node B.

- 1. Before beginning this testing, ensure the following:
 - The failover script you are testing is installed.
 - The configuration file (/var/ha/ha.conf) includes blocks for the application whose script you are testing.
 - IRIS FailSafe is not running on the cluster.
 - The application you are testing is not running on either node in the cluster.
- 2. Start up IRIS FailSafe and the applications whose script you are testing by entering these commands on both nodes:
 - # chkconfig failsafe on
 # /etc/init.d/failsafe start
- 3. Wait until both nodes reach normal state. You can verify this using this command on each node:

```
# /usr/etc/ha_admin -i
ha_admin: Node controller state normal
```

- 4. Verify that Node A and Node B are running the instances of all applications for which they are the primary node.
- 5. On Node A, enter this command:
 - # /usr/etc/ha_admin -s
- 6. Verify that no highly available applications are running on Node A, and that all instances of the highly available applications are running on Node B. Node A must be in standby state and Node B must be in degraded state.
- 7. On Node A, enter this command:
 - # /usr/etc/ha_admin -fr
- 8. Verify that Node A and Node B are running the instances of all applications for which they are the primary node and both nodes are in normal state.

- 9. On Node B, enter this command:
 - # /usr/etc/ha_admin -s
- 10. Verify that no highly available applications are running on Node B, and that all instances of the highly available applications are running on Node A. Node B must be in standby state and Node A must be in degraded state.
- 11. On Node B, enter this command:
 - # /usr/etc/ha_admin -fr
- 12. Verify that Node A and Node B are running the instances of all applications for which they are the primary node and both nodes are in normal state.

Names Used in Template Configuration Files

This appendix describes each of the names—block names, section names, and parameter names—used in the sample configuration files included with the IRIS FailSafe product and the IRIS FailSafe options.

When you are developing scripts for failing over a new highly available service, you can use these names (for ease of maintenance you should use them for the purpose described here) or you can define new names. Defining new names is described in Chapter 2, "Modifying the Configuration File for a New Highly Available Service."

The sections in this appendix are as follows:

- "Block Names" on page 76
- "Section Names" on page 77
- "Parameter Names" on page 78

Block Names

Table A-1 lists the blocks in configuration file templates included in IRIS FailSafe products and summarizes their contents.

 Table A-1
 Major Blocks in the Configuration File

Name	Description		
action appclass	Describes the scripts that are to be executed for the highly available service <i>appclass</i> . An action block exists for each highly available service. For all highly available services the action block specifies the local and monitor scripts if used. For the "main" highly available service, it also specifies the <i>giveaway</i> , <i>giveback</i> , <i>takeback</i> , <i>takeover</i> , and <i>kill</i> scripts.		
action-timer appclass	Describes the various timers that are used by the application monitor to decide when to execute and time out a monitoring script. The values can be adjusted based upon the expected response times of instances of the highly available service <i>appclass</i> .		
application-class appclass	Describes one highly available service that is failed over by this IRIS FailSafe cluster. These blocks identify the nodes that provide the highly available service <i>appclass</i> in normal state.		
filesystem label	Each filesystem block describes a single filesystem on a shared disk. For each filesystem, it specifies the primary and backup nodes and mount information. There should be one filesystem block for each filesystem on a shared disk in the cluster.		
informix label	An informix section is present for each INFORMIX database failed over in this cluster.		
interface-pair label	Contains IP addresses to be failed over and the primary interface and backup interface for the IP addresses.		
internal	Describes the various timeout values that are used by IRIS FailSafe daemons. The values in this block, except for long-timeout, must no be changed.		
nfs label	nfs blocks are present if NFS is failed over in this cluster. Each nfs block describes the NFS export information associated with a single exported filesystem.		
node label	Each node block describes the network interface configuration and heartbeat and serial information for a node in the cluster.		

Table A-1 (continued)	Major Blocks in the Configuration File		
Name	Description		
oracle label	An oracle section is present for each Oracle database failed over in this cluster.		
sybase label	A sybase section is present for each Sybase Server failed over in this cluster.		
system-configuration	Describes global variables for the IRIS FailSafe cluster as a whole.		
volume <i>label</i>	Each volume block describes the ownership and location of one XLV volume.		
webserver label	webserver blocks are present if any of the nodes in this cluster are Netscape servers. Each webserver block describes the Netscape configurations of one node.		

Section Names

Table A-2 lists the section names in the configuration file templates.

 Table A-2
 Section Names in Template Configuration Files

Section Name	Description
heartbeat	This section specifies heartbeat monitoring parameters.
interface label	A node network interface that is to be monitored. The interface label is created from the ip-address and name parameters and must be unique in the configuration file. There is one interface section for each public interface in the node that is part of IRIS FailSafe. Not all public interfaces need to be part of IRIS FailSafe.
mount-info	Contains filesystem mounting information.
web-confign	Describes one Web server instance on a node.

Parameter Names

Table A-3 lists the parameters used in the configuration file templates provided with IRIS FailSafe software options. Additional parameters can be defined as needed when failover of other applications is added.

 Table A-3
 Parameters in Template Configuration Files

Parameter Name	Possible Values	Description
agent	pathname	The pathname of the agent for the highly available service.
backup-node	label	nodename is a name returned by hostname.
		For volume blocks: The backup node for this XLV logical volume. The value assigned to backup-node must match the label for a node block.
		For webserver blocks: The backup node for the Netscape server. The value must match the label for a node block.
backup-server	string	The name of the Backup Server for this SQL Server.
broadcast-addr	X.X.X.X	For node blocks: The broadcast address for the subnet.
		For interface-pair blocks: The subnet broadcast IP address for the IP aliases in <i>X.X.X.X</i> notation.
config-file	string	The INFORMIX configuration file for this node. Its value is the value of the ONCONFIG environment variable.
controlled-failback	true false (not set)	Controls whether this node automatically moves to normal state after a failure. If controlled-failback is set to true , the node doesn't move to normal state after failure; it moves to controlled failback state. If set to false or not set, the node moves to normal state.
db-avail	high low	If the value is high (the default value), a database server (INFORMIX, Oracle, or Sybase) failure forces a failover. If the value is low , a failure of the database server doesn't force a failover, but the failover is reported.

 Table A-3 (continued)
 Parameters in Template Configuration Files

		1 0
Parameter Name	Possible Values	Description
db-probe-time	integer	Specifies the length of time (in seconds) between the completion of one probe of the database by the database agent and the beginning of the next probe.
db-shutdown- timeout	integer	The timeout for the Oracle shutdown script specified by shutdown-script. If the shutdown script doesn't complete in this many seconds, IRIS FailSafe performs an abort shutdown of the Oracle instance.
db-retry-count	integer	The number of monitoring retries by the database agent before a failure is declared.
db-timeout	integer	Defines (in seconds) the time the database agent waits for a response to its probe from the database instance.
devname	pathname	The device filename for the XLV logical volume.
devname-group	string	The group of the device name for the XLV logical volume (reported by <i>ls -l</i>). The default value is sys .
devname-mode	fs_mode	The access mode of the device name for the XLV logical volume (reported by <i>ls -l</i>). The default value is 0600 .
devname-owner	string	The user ID of owner of the device name for the XLV logical volume (reported by <i>ls -l</i>). The default value is root .
export-info	export_mode	Filesystem export options (see the exports(4) reference page and the section "Wsync Filesystem Options" in Chapter 4 of the IRIS FailSafe Administrator's Guide).
export-point	pathname	The pathname of an exported filesystem.
filesystem	label	A filesystem to be exported. The value of this parameter must match the label of a filesystem block and the label of the block.
fs-type	xfs	The filesystem type. Only xfs filesystems are supported.
giveaway	pathname	The pathname of the <i>giveaway</i> script in /var/ha/actions.

 Table A-3 (continued)
 Parameters in Template Configuration Files

Parameter Name	Possible Values	Description
giveback	pathname	The pathname of the <i>giveback</i> script in /var/ha/actions.
hb-lost-count	integer	Specifies how many heartbeat probe failures must occur to declare a heartbeat failure. The recommended value is 3.
hb-private-ipname	string X.X.X.X	This node's IP address for the private network used by heartbeat and control messages.
hb-probe-time	integer	Heartbeat messages begin this many seconds after the node controller tells the application monitor to start monitoring. Also, this value specifies how long to wait (in seconds) after completion of the last heartbeat message to begin the next heartbeat message. The recommended value is 5.
hb-public-ipname	hostname X.X.X.X	This node's IP address for the public network that is used for heartbeat messages if the private network fails. This IP address is a fixed IP address.
hb-timeout	integer	Specifies how long (in seconds) to wait for a heartbeat response before declaring a failure. The recommended value is 5.
httpd-dir	pathname-port	The Netscape [®] server root location.
httpd-options-file	string	The Netscape configuration file that starts multiple Netscape servers. The value is not a full pathname; it is a file in the directory <code>/etc/config</code> . The default value is <code>ns_httpd.options</code> , which is the configuration file for the Netscape Communications server.
httpd-restart	false true	If the two nodes have identical Netscape server configurations (a dual-active configuration with the same configuration information, log locations, and document root), then the Netscape server doesn't need to be restarted after failover (because an identical server is already running) and httpd-restart should be set to true . Otherwise, the Netscape server needs to be started on the backup node after failover and httpd-restart should be set to false .

 Table A-3 (continued)
 Parameters in Template Configuration Files

Parameter Name	Possible Values	Description
httpd-script	pathname	The full pathname of the script used to start and stop the Netscape server. The default value is /etc/init.d/ns_httpd, which is the configuration file for the Netscape Communications server.
instance-id	string	The value of \$ORACLE_SID for the Oracle instance that IRIS FailSafe is monitoring.
interface-probe- interval	integer	The length of time (in seconds) between the completion of one probe of the local interfaces and the beginning of the next probe. The value is rounded to the nearest five-second increment.
interface-probe- timeout	integer	The length of time (in seconds) that the interface agent waits after probing the local interfaces without a response before declaring a failure.
ip-address	string X.X.X.X	For node blocks: The fixed IP address of this network interface. It can be a name (<i>string</i>) or an address (<i>X.X.X.X</i>).
		For nfs blocks: One (any one) of the IP addresses or IP aliases on the node that is primary for this filesystem, preferably in the form X.X.X.X. A good choice is the IP address or IP alias used by clients to mount the filesystem. If an IP name is used, the length of time required to resolve the name to an address could require that the lmon-timeout value for NFS be increased.
		For webserver blocks: The high availability IP address used by clients to access the Netscape server. An IP address of the form <i>X.X.X.X</i> is recommended.
ip-aliases	(string) (str1 str2 str3)	For interface-pair blocks: Specifies the list of IP addresses to be failed over using IP aliasing.
		For application-class nfs blocks: Specifies a list of IP aliases. This parameter, with at least one IP alias, is required if NFS file locking is used (listing at least one IP aliases is recommended). Each IP alias that is used for NFS should be listed.

 Table A-3 (continued)
 Parameters in Template Configuration Files

Parameter Name	Possible Values	Description
kill	/usr/etc/ha_kill	The pathname of the <i>ha_kill</i> command.
lmon-probe-time	integer	The probe interval (in seconds) for local monitoring of this highly available service.
lmon-timeout	integer	Local monitoring of this highly available service times out in <i>integer</i> seconds if no response is received.
local-monitor	pathname	The pathname of the local monitoring script for this highly available service.
long-timeout	integer	The maximum time taken by the <i>takeover</i> , <i>takeback</i> , <i>giveaway</i> , and <i>giveback</i> scripts. If these scripts cannot be executed in this length of time, the value should be increased. This value is also used as the maximum time taken by several types of internal IRIS FailSafe communications.
MAC-address	X:X:X:X:X:X	MAC addresses are required only if the network interfaces have to use MAC address impersonation. See the section "Network Interfaces and IP Addresses" in Chapter 1 of the IRIS FailSafe Administrator's Guide.
mail-dest-addr	user@host (not set)	Mail is sent to this address when private network failure has been detected, the local node controller process appears to be hung or dead, the cluster is transitioning to degraded state, the cluster is transitioning to standby state, killing of a node fails, the <i>ha_killd</i> daemon has died, the <i>ha_killd</i> daemon could not be started, or the reset device monitor has failed. Do not set if mail is not configured on this node.
master-db-fs	label	<i>label</i> is the filesystem label for the filesystem of the master database.
master-db-vol	label	<i>label</i> is the volume label for the volume of the master database.
mode	fs_mode	Filesystem mount options (see the fstab(4) reference page) and the section "Wsync Filesystem Options" in Chapter 4 of the IRIS FailSafe Administrator's Guide.

 Table A-3 (continued)
 Parameters in Template Configuration Files

Parameter Name	Possible Values	Description
monitoring-level	1 2	For informix blocks: Defines which test is done to determine if INFORMIX is up. If the value is 1, the database agent executes the <i>onstat</i> command and searches the output for the pattern specified by running-indicator-strings. If there is a match, the database is up. If the value is 2, the database agent uses a database call to determine if INFORMIX is up
		For webserver blocks: Defines which test is done to determine if Netscape is up. If the value is 1, the ps command is used to check if the <i>httpd</i> process is running on the node. If the value is 2, an http request is sent to the <i>httpd</i> process to determine if the Netscape server is running. The value 2 is a stricter check. If the value 1 is specified, the parameter search-string in the webserver block is used.
mount-point	pathname	The pathname of a filesystem mount point. Both nodes use the same mount point.
name	interface	<i>interface</i> is a network interface. Each node has several network interfaces. For example, a CHALLENGE S node has the network interfaces ec0, ec2, and ec3.
netmask	X.X.X.X	For node blocks: The netmask used to identify this node on the subnet.
		For interface blocks: The netmask for the IP aliases.
port-num	integer	The Netscape server port number.
primary-interface	label	A name for an interface on which the IP aliases are configured in normal state, typically created by combining the hostname and interface name. The value must match an interface section label in a node block.

 Table A-3 (continued)
 Parameters in Template Configuration Files

Parameter Name	Possible Values	Description
pwrfail	true false (not set)	This parameter does not apply to CHALLENGE S nodes. When set to true (the default), it allows the surviving node to attempt to go to degraded state after it detects a power failure on the other node (or the private network, public network, and serial connections are broken). If pwrfail is set to false or not set, the node goes to standby state after it detects a power failure on the other node.
re-mac	true false (not set)	If the IP address and the physical address of the primary interface are to be transferred to the backup interface when a failover occurs, set re-mac to true . Otherwise set it to false or leave it undefined.
release-dir	pathname	For informix blocks: The INFORMIX release directory specified in the INFORMIX configuration. This value is the value of the environment variable INFORMIXDIR.
		For Sybase blocks: The Sybase release directory specified in the Sybase configuration, which is also the value of the Sybase environment variable.
remote-send-probe- interval	integer	The frequency (in seconds) of messages to the interface agent on the other node. This parameter must be less than the value of interface-probe-interval, but should be only slightly less. The value is rounded to the nearest five seconds.
remote-send-timeout	integer	The length of time (in seconds) between the completion of one probe of the other node's highly available interfaces and the beginning of the next probe. This parameter must be less than the value of interface-probe-interval, but should be only slightly less. The value is rounded to the nearest five seconds.

 Table A-3 (continued)
 Parameters in Template Configuration Files

Parameter Name	Possible Values	Description		
reset-host	hostname	Applies only to nodes running the Silicon Graphics Oracle Parallel Server product. If an IRISconsole is used to provide reset functionality, <i>hostname</i> is the hostname of the Indy running IRISconsole software. reset-host is ignored if reset-tty is set. Do not set reset-host if you are not running OPS in the IRIS FailSafe cluster.		
reset-tty	serial_devicename	The device filename of the serial port on this node that is used by the serial cable connected to the system controller port on the other node or to the remote power control unit.		
retry-count	integer	Determines the number of retries done by the local monitoring script. The application monitor declares a local monitor failure after lmon-timeout seconds independent of the retry-count value.		
running-indicator- strings	string	A string that is used as a search pattern in determining if On-Line is up. The output of the <i>onstat</i> command is searched. The string must begin and end with double-quotes. An OR symbol () can be used in the string to separate multiple search patterns. The string cannot contain any blanks and the search is case sensitive.		
sa-passwd	string	The unencrypted password of the database system administrator (sa-user). This parameter should be omitted if there is no password.		
sa-user	string	For informix blocks: The INFORMIX login name of the INFORMIX database system administrator.		
		For sybase blocks: The Sybase login name of the Sybase database system administrator.		
the ru rec		Specifies the string to be searched for in the output of the <i>ps</i> command to verify that the <i>httpd</i> process is running, for example ns-httpd. This parameter is required if the monitoring-level parameter in a webserver block has the value 1.		

 Table A-3 (continued)
 Parameters in Template Configuration Files

Parameter Name	Possible Values	Description A name for the backup interface, an interface on the other node that replaces the primary interface on failover. It is typically created by concatenating the hostname and interface name. The value must match an interface section label in a node block.		
secondary-interface	label			
server-node	label	For application-class blocks: The primary node for instances of the highly available service. Server-node is listed twice if each of the nodes in a cluster serves as the primary node for some instances. The values assigned to server-node must match the labels for node blocks.		
		For volume blocks: The primary node for this XLV logical volume. The value assigned to server-node must match the label for a node block.		
		For webserver blocks: The primary node for this Netscape server. The value must match the label for a node block.		
short-timeout	integer	The maximum length of time (in seconds) for certain IRIS FailSafe internal communications tasks to complete. Do not change this value.		
shutdown-script	pathname	The pathname of a custom Oracle shutdown script that replaces the standard Oracle shutdown script.		
shutdown-options	integer	Defines how the database is shut down on a failover. The possible values are: 0—normal shutdown, 1— shutdown with nowait and no checkpointing, and 2—shutdown with nowait with checkpointing.		

 Table A-3 (continued)
 Parameters in Template Configuration Files

Parameter Name	Possible Values	Description		
start-monitor-time	integer	For application-class blocks: This parameter specifies how long (in seconds) after a node releases this highly available service (giveaway) that the application monitor waits before starting monitoring of the highly available service. It is required.		
		For an action-timer block for a highly available service that has an agent (for example interfaces, sybase, informix, or oracle): The length of time (in seconds) that the interface agent waits after it is told by the application monitor to start monitoring the local interfaces before beginning to monitor. This wait ensures that the database instance has had time to start up and must be greater than or equal to the value of long-timeout.		
		For an action-timer block for a highly available service that doesn't have an agent: The length of time (in seconds) that the monitoring script waits after it is told by the application monitor to start monitoring before it begins monitoring. The value must be greater than or equal to the value of long-timeout and is required.		
startup-script	pathname	The pathname of a custom Oracle startup script that replaces the standard Oracle startup script.		
statmon-dir	pathname	Specifies the pathname of a directory on a shared filesystem belonging to its server-node where NFS locking information is stored. The basename of this directory must be <i>statmon</i> .		
takeback	pathname	The pathname of the takeback script in /var/ha/actions.		
takeover	pathname	The pathname of the takeover script in /var/ha/actions.		
unix-user	login_id	The IRIX login name of the owner of the release directory for the database software.		
version-major	1	A version number that, along with version-minor, specifies the configuration file format used in this file. It is 1 for IRIS FailSafe 1.0 and IRIS FailSafe 1.1.		

 Table A-3 (continued)
 Parameters in Template Configuration Files

Parameter Name	Possible Values	Description		
version-minor	0 1	A version number that, along with version-major, specifies the configuration file format used in this file. It is 0 for IRIS FailSafe 1.0 configuration files (which can run on IRIS FailSafe 1.1 as well) and 1 for IRIS FailSafe 1.1 configuration files.		
volume-name	label	volname must match a volume block label.		
webserver-num	integer	The number of Netscape servers configured on this node (server-node). This is the number of web-configured sections for this Netscape server.		

Tell Us About This Manual

As a user of Silicon Graphics products, you can help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please send the title and part number of the document with your comments. The part number for this document is 007-3298-001.

Thank you!

Three Ways to Reach Us

- To send your comments by electronic mail, use either of these addresses:
 - On the Internet: techpubs@sgi.com
 - For UUCP mail (through any backbone site): [your_site]!sgi!techpubs
- To **fax** your comments (or annotated copies of manual pages), use this fax number: 415-965-0964
- To send your comments by **traditional mail**, use this address:

Technical Publications
Silicon Graphics, Inc.
2011 North Shoreline Boulevard, M/S 535
Mountain View, California 94043-1389