Sun Cluster 2.2 API Developer's Guide

901 San Antonio Road
Palo Alto, , CA 94303-4900
USA 650 960-1300 Fax 650 969-9131

Part Number 806-5344
August 2000, Revision A

Copyright Copyright 2000 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303-4900 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd. For Netscape Communicator™, the following notice applies: (c) Copyright 1995 Netscape Communications Corporation. All rights reserved.

Sun, Sun Microsystems, the Sun logo, AnswerBook2, docs.sun.com, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Federal Acquisitions: Commercial Software—Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2000 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, Californie 94303-4900 U.S.A. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd. La notice suivante est applicable à Netscape Communicator™: (c) Copyright 1995 Netscape Communications Corporation. Tous droits réservés.

Sun, Sun Microsystems, le logo Sun, AnswerBook2, docs.sun.com, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.

Adobe PostScript

Please
Recycle

Contents

Preface

Sun™ Cluster 2.2 is a software product that supports specific two to four node server hardware configurations. It is compatible with the Solaris™ 2.6, Solaris 7, and Solaris 8 operating environments. When configured properly, the hardware and software together provide highly available data services. Sun Cluster depends upon the mirroring, diskset capabilities, and other functionality provided by a volume manager. Sun Cluster supports two volume managers, Solstice™ DiskSuite and VERITAS Volume Manager (VxVM).

Sun Cluster provides two application programming interfaces (APIs) for making data services highly available. The data services API permits client-server data services to be layered on top of the Sun Cluster software. The fault monitor API enables programmers to develop fault monitors for a new highly available (HA) data service.

Usually, the data service of interest is one that already exists and was developed in a non-HA environment. This API was designed to permit an existing data service to be added easily to the Sun Cluster environment. This manual provides tips on how to achieve the addition.

The *Sun Cluster 2.2 API Developer's Guide* describes the usage of the two APIs. It discusses conventions that a data service should follow to be highly available.

This book includes an example for making a data service highly available. The example data service is the Internet Domain Name Service (DNS), specifically Sun"s implementation, which is the daemon program `in.named(1M)`. The `in.named` example is presented for illustrative purposes only; running the presented code is not supported by Sun.

This manual should be used with the other hardware and software books listed under "Related Documentation" on page 6, and with the man pages associated with the API. These man pages are: `hareg(1M)`, `haget(1M)`, `hads(1M)`, `ha_open(1M)`, `ha_get_calls(1M)`, `hactl(1M)`, `hatimerun(1M)`, `rpc.pmfd(1M)`, `pmfadm(1M)`, and `halockrun(1M)`. Of particular interest is the *Sun Cluster 2.2*

Software Installation Guide, which describes the cluster environment into which you will integrate your data service.

Who Should Use This Book

This book is intended for programmers responsible for integrating an existing data service application into the HA environment. The instructions and discussions are intended for a technically advanced audience.

The instructions in this book assume the reader has a high level of expertise with the data service being integrated.

Related Documentation

The documents in Table P–1 contain helpful information for system administrators and service providers.

TABLE P–1 List of Related Documentation

Product Family	Title	Part Number
Sun Cluster	*Sun Cluster 2.2 System Administration Guide*	806-4649
	Sun Cluster 2.2 Software Installation Guide	806-4648
	Sun Cluster 2.2 Error Messages Manual	805-4242
	Sun Cluster 2.2 Release Notes	806-4652
	Sun Cluster 2.2 Hardware Site Preparation, Planning, and Installation Guide	806-4748
	Sun Cluster 2.2 Hardware Service Manual	806-4749
Solstice DiskSuite	*Solstice DiskSuite 4.2 Installation/Product Notes*	805-5960

TABLE P–1 List of Related Documentation *(continued)*

Product Family	Title	Part Number
	Solstice DiskSuite 4.2 User's Guide	805-5961
	Solstice DiskSuite 4.2 Reference	805-5962
	Solstice DiskSuite 4.2.1 Reference Guide	806-3204
	Solstice DiskSuite 4.2.1 User's Guide	806-3205
	Solstice DiskSuite 4.2.1 Installation and Product Notes	806-3206
VERITAS Volume Manager	*Veritas Volume Manager for Solaris*	100-001125A (VERITAS)
Oracle	*Oracle Installation Guide for Sun SPARC Solaris 2.x*	802-6994
Name Services	*Name Services Administration Guide*	801-6633
	Name Services Configuration Guide	801-6635
Other Manuals	*NFS Administration Guide*	801-6634
	TCP/IP Network Administration Guide	801-6632

Typographic Conventions

TABLE P–2 Typographic Conventions

Typeface or Symbol	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output.	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`

`AaBbCc123`	What you type, when contrasted with on-screen computer output.	`%` **`su`**
		`Password:`

AaBbCc123	Book titles, new words or terms, words to be emphasized. Command-line variable; replace with a real name or value.	Read Chapter 6 in the *User's Guide.* These are called *class* options. You *must* be `root` to do this. To delete a file, type `rm` *filename*.

Shell Prompts

TABLE P–3 Shell Prompts

Shell	Prompt
C shell	*machine_name*%
C shell superuser	*machine_name*#
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell superuser	`#`

CHAPTER 1

Data Services API

This chapter introduces the Sun Cluster Data Services API and the concepts needed to make your data service applications highly available. There is also a section on the differences between the API implementation on Solstice HA 1.3 and Sun Cluster 2.x

Overview

The Sun Cluster Data Service API uses command-line utilities and a C-callable library. For convenience, all C-callable functionality is also available using the command-line utility programs. This enables you to code in a scripting language such as the Bourne shell (`sh(1)`), if you choose.

The API is defined by its man pages:

- `hareg(1M)`—Controls registration and activation of Sun Cluster data services
- `haget(1M)`—Queries current state of Sun Cluster configuration
- `hads(3HA)`—Library routines for Sun Cluster data services
- `ha_open(3HA)`, `ha_close(3HA)`—Sun Cluster environment open and close
- `ha_get_calls(3HA)`—Gets Sun Cluster environment
- `hatimerun(1M)`—Provides a convenient facility for timing out the execution of another child program. It is useful when programming in scripting languages such as the Bourne shell.
- `halockrun(1M)`—Provides a convenient means to claim a file lock on a file and run a program while holding that lock. It is useful when programming in scripting languages such as the Bourne shell.
- `hactl(1M)`—Provides various control operations on Sun Cluster. The control operations include requesting the movement of a logical host from one physical

host to another (possibly forcibly), requesting the movement of all logical hosts that a physical host currently masters to other physical host(s), and requesting a cluster reconfiguration.

- `pmfadm(1M)`—Provides the administrative, command line interface, to the process monitor facility.
- `rpc.pmfd(1M)`—RPC-based server for serving the process monitor facility that is used by the Sun Cluster 2.2 System.

The command line utilities and the C-callable library are documented in the man pages.

Interaction Between Data Services and the Sun Cluster Software

When a data service first registers with Sun Cluster, it registers a set of call-back programs, or *methods*. Sun Cluster makes call-backs to the data service"s methods when certain key events occur in the cluster. The remainder of this section describes the three basic methods required to make any data service run in the Sun Cluster environment. The methods are `start`, `stop`, and `abort`.

After the failure of a host, Sun Cluster itself moves the logical host (both its diskset and its logical network IP addresses) to one of the surviving hosts. At this point, the data service"s software must be restarted on the surviving host. Sun Cluster itself cannot restart a data service. Instead, it makes a call to the data service telling it to restart itself. This call is to the data service"s `start` or `start_net` method.

The Sun Cluster `haswitch(1M)` command smoothly shuts down a logical host on one physical server in preparation for moving the logical host to another physical server. For Sun Cluster to coordinate this shut-down work with layered data services, each data service also registers a `stop` method. Sun Cluster calls the data service"s `stop` method during `scadmin switch` or `haswitch(1M)` operations, and whenever Sun Cluster is stopped using `scadmin stopnode`. This stop method performs a smooth, safe shutdown of the data service. This occurs without waiting for clients on the network to completely finish their work, because waiting for a client could introduce an unbounded delay.

Sun Cluster continuously monitors the health of the physical servers in the cluster. In some cases, Sun Cluster will decide that a physical server is failing, but is still able to execute some “last wishes” cleanup code before Sun Cluster halts and reboots the server. In this case, each data service is given an opportunity to execute last wishes cleanup code before Sun Cluster halts the server. Sun Cluster does this by calling the `abort_net` method of each data service. A data service that does not need or want the last wishes cleanup opportunity can choose not to register an `abort` method.

Logical Host Configuration Issues

A data service is made highly available by exploiting the Sun Cluster concept of a *logical host*. The data service's data is placed on a logical host's *diskset*. A diskset is dual-ported, making the data accessible by a surviving server in the event that one server fails. For network access by clients on the network, the data service advertises the logical host name as the server name that clients should use. A logical network IP address *failover* causes network clients of the data service to move with the logical host.

Data Service Use of Single or Multiple Logical Hosts

In Sun Cluster, there can be any number of logical hosts, so your data service implementation should not depend on a certain quantity. You must decide whether your data service will keep its data in just one or in multiple logical hosts.

Generally, it is easier to design and implement a data service that uses just one logical host. In that case, all of the data service's data is placed only on that logical host's diskset. The data service needs just one set of daemons. A physical host runs the daemons for that data service only if the physical host currently masters the single logical host that the data service uses. When the physical host takes over mastery of the logical host, the data service's `start` method can start up the daemons. When the physical host is giving up mastery of the logical host, the data service's `stop` method can stop the daemons. In some cases, killing the daemons by sending a kill signal will suffice.

If you use multiple logical hosts, you must be able to split the data service's data into disjoint sets. The sets must be split so that no operation the data service needs to perform requires data from more than one set.

Consider Sun's HA-NFS product, which has multiple file systems with different data^{TM} residing in each file system. For HA-NFS, each logical host has its own set of NFS file systems. Each physical host NFS shares the file systems that belong to the logical hosts that it masters. The sets of NFS file systems belonging to the two logical hosts are disjoint.

Using multiple logical hosts enables some rudimentary load balancing: when both physical hosts are up, each physical host can master one of the logical hosts and handles the data service"s traffic for that logical host. Thus, both physical hosts are doing useful work in addition to acting as standbys for each other.

For some data services, splitting the data into disjoint collections such that no data service operation requires more than one collection is not feasible. The `in.named` example described in Chapter 2 "Sample Data Service", is such a data service. It has only one set of interdependent data files, and it would be difficult to split them into disjoint sets.

Note - Configure the data service to use just one of the logical hosts, unless the data is easily split into disjoint collections *and* there is significant benefit to the rudimentary load balancing enabled by use of multiple logical hosts.

Required File System for Each Logical Host

Each Sun Cluster logical host has at least one diskset containing one or more file systems or raw partitions. Sun Cluster requires that each logical host has one file system that is special, in that it must exist and must have a particular name (that is, it must be mounted on a particular directory name in the name space hierarchy). When Sun Cluster is first installed and configured, the `scconf(1M)` program assists the administrator in creating the required file system, thus following the required convention. Sun Cluster uses the term administrative file system to refer to this special required file system.

Required Administrative File System Conventions

If your data service uses the administrative file system, it must adhere to the conventions described in this section.

Per Data-Service Subdirectory

Each data service should place its administrative data in its own subdirectory of the administrative file system. For example, if the data service uses Solaris packages, then the subdirectory should have a name of the form /*administrative_file_system*/ *PkgName,* where *PkgName* is the name of your data service package.

If the package mechanism is not used, then the data service should use the same name that it supplied as its data service name when it registered with Sun Cluster using `hareg(1M)`. The `hareg(1M)` utility detects and prohibits naming conflicts. If your implementation uses logical host “hahost1,” and calls `hareg(1M)` with the name “hainnamed,” you create the administrative subdirectory /`hahost1`/ `hainnamed.`

Small Amount of Data

The administrative file system is relatively small. Each data service should limit the amount of administrative data it keeps in the administrative file system to a few kilobytes. If a larger amount of administrative data is required, use the administrative file system to point at another directory in one of the logical host’s file systems. The data service"s user data should not be stored in the administrative file system, because for most data services, that data would be too large.

Data Service Requirements

The following sections present the requirements that a data service must meet to participate in the Sun Cluster Data Service API.

Client-Server Environment

Sun Cluster is designed for client-server networking environments. Sun Cluster cannot operate in time-sharing environments in which applications are run on a server that is accessed through `telnet` or `rlogin`. Such models typically have no inherent ability to recover from a server crash.

Crash Tolerance

The data service must be crash-tolerant. This means that the data service’s daemon processes must be relatively stateless, in that they write all updates to disk synchronously.

When a physical host that masters a logical host crashes and a new physical host takes over, Sun Cluster calls the `start` method of each data service. The `start` method triggers any crash recovery of the on-disk data. For example, if the data service uses logging techniques, the `start` method should cause the data service to carry out crash recovery using the log.

Multihosted Data

The logical host’s disksets are multihosted so that when a physical host crashes, one of the surviving hosts can access the disk. For a data service to be highly available, its data must be highly available, and thus its data must reside on the logical host"s diskset.

A data service might have command-line switches or configuration files pointing to the location of the data files. If the data service uses hard-wired path names, it might be possible to change the path name to a symbolic link that points to a file in the logical host’s diskset, without changing the data service code. See Appendix A, for a more detailed discussion about using symbolic links.

In the worst case, the data service's code must be modified to provide some mechanism for pointing to the actual data location. You can do this by implementing additional command-line switches.

Sun Cluster supports the use of UFS, VxFS, and raw partitions on the logical host's diskset. When the system administrator installs and configures Sun Cluster, he or she must specify which disk resources to use for UFS or VxFS file systems and which for raw partitions. Typically, raw partitions are used only by database servers and multimedia servers.

Host Names

You must determine whether the data service ever needs to know the host name of the server on which it is running. If so, the data service might need to be modified to use the host name of the logical host, rather than that of the physical host. Recall that the Sun Cluster concept of "logical host" involves having a physical host "impersonate" a logical host's host name and IP address.

Occasionally, in the client-server protocol for a data service, the server returns its own host name to the client as part of the contents of a message to the client. For such protocols, the client could be depending on this returned host name as the host name to use when contacting the server. For the returned host name to be usable after a takeover or switchover, the host name should be that of the logical host, not the physical host. In this case, you must modify the data service code to return the logical host name to the client.

Multihomed Hosts

The term multihomed host describes a host that is on more than one public network. Such a host has multiple host names and IP addresses; it has one host name/IP address pair for each network. Sun Cluster is designed to permit a host to appear on any number of networks, including just one (the non-multihomed case). Just as the physical host name has multiple host name/IP address pairs, each logical host has multiple host name/IP address pairs, one for each public network. By convention, one of the host names in the set of pairs is the same name as that of the logical host itself. When Sun Cluster moves a logical host from one physical host to another, the complete set of host name/IP address pairs for that logical host is moved.

For each Sun Cluster logical host, the set of host name/IP address pairs is part of the Sun Cluster configuration data and is specified by the system administrator when Sun Cluster is first installed and configured. The Sun Cluster Data Service API contains facilities for querying the set of pairs, specifically, the `names_on_subnets` field described in the `hads(3HA)` and `haget(1M)` man pages.

Most off-the-shelf data service daemons that have been written for Solaris already handle multihomed hosts properly. Many data services do all their network communication by binding to the Solaris wildcard address INADDR_ANY. This automatically causes them to handle all the IP addresses for all the network interfaces. INADDR_ANY effectively binds to all IP addresses currently configured on the machine. A data service daemon that uses INADDR_ANY generally does not have to be changed to handle the Sun Cluster logical host's IP addresses.

Binding to INADDR_ANY Versus Binding to Specific IP Addresses

Even in the non-multihomed case, the Sun Cluster logical host concept allows the machine to have more than one IP address. It has one for its own physical host and one additional IP address for each logical host it currently masters. When a machine becomes the master of a logical host, it dynamically acquires an additional IP address. When it gives up mastery of a logical host, it dynamically relinquishes an IP address.

Some data services cannot work properly using only INADDR_ANY. These data services must dynamically change the set of IP addresses to which they are bound as a logical host is mastered or unmastered. The starting and stopping methods provide the hooks for Sun Cluster to inform the data service that a logical host has appeared or disappeared. One strategy for such a data service to accomplish the rebinding is for its stop and start methods to kill and restart the data service's daemons.

During cluster reconfiguration, there is a relationship between the order in which data service methods are called and the time when the logical host's network addresses are configured by Sun Cluster. See the hareg(1M) man page for details about this relationship.

By the time the data service's stop method returns, the data service should have stopped using the logical host's IP addresses. Similarly, by the time the start_net method returns, the data service should have started to use the logical host's IP addresses. If the data service uses INADDR_ANY rather than binding to individual IP addresses, then there is no problem. If the data service's stop and start methods accomplish their work by killing and restarting the data service's daemons, then the data service stops and starts using the network addresses at the appropriate times.

Client Retry

To a network client, a takeover or switchover appears to be a crash of the logical host followed by a fast reboot. Ideally, the client application and the client-server protocol are structured to do some amount of retrying. If the application and protocol already handle the case of a single server crashing and rebooting, then they also will handle the case of the logical host being taken over or switched over. Some applications might elect to retry endlessly. More sophisticated applications notify the user that a long retry is in progress and allow the user to choose whether or not to continue.

Registering a Data Service

A data service is registered with Sun Cluster using the `hareg(1M)` program. Registration is persistent in that it survives across takeovers, switchovers, and reboots. Registration with Sun Cluster is usually done as the last step of installing and configuring a data service. Registration is a one-time event. A data service also can be unregistered with `hareg(1M)`. See the `hareg(1M)` man page for details.

In addition to the distinction between registered versus unregistered, Sun Cluster has the concept of a data service being either "on" or "off." The purpose of the "on" or "off" state is to provide the system administrator with a mechanism for temporarily shutting down a data service without taking the more drastic step of unregistering it. For example, a system administrator can turn a data service "off" to do stand-alone backups. While the data service is "off," it is not providing service to clients. When a data service is "off," the parameters that Sun Cluster passes to the data service's methods indicate that the data service should not be servicing data from any of the logical hosts.

When a data service is first registered with Sun Cluster, its initial state is "off." The `hareg(1M)` program is used to transition a data service between the "off" and "on" states. The work of moving a data service between states is accomplished through a reconfiguration as described in the `hareg(1M)` man page.

Before unregistering a data service, the system administrator first must transition the data service into the "off" state by calling `hareg(1M)`.

Differences Between the Solstice HA 1.3 and Sun Cluster 2.x API

In Solstice HA, the `hareg(1M)` man page defined an explicit reconfiguration sequence, for example, that `stop` methods are called before `start` methods are called, and that when a `stop` method is called a `start` method is also eventually called.

However, the Sun Cluster 2.x implementation deviates from the Solstice HA model. Most notably, you should not rely on the overall reconfiguration sequence too much. In Sun Cluster 2.x, it is possible for the following to occur:

- When moving a logical host off of a physical host, stop_net and `stop` methods will be called, however, the start and `start_net` methods will not necessarily be called. This deviates from `hareg(1M)` man page.

- When moving a logical host onto a physical host, start and `start_net` methods are called. However, the `stop_net` and `stop` methods are not necessarily called. Again, this deviates from the `hareg(1M)` man page.
- Solstice HA 1.3 has the behavior that there is a single call to each method for all the logical hosts that are moving during a particular reconfiguration. For example, let us say that a physical host is mastering two logical hosts and that physical host crashes. Both of the logical hosts will need to move to the surviving physical host (consider just a two node cluster for now). In Solstice HA 1.3, the methods of each registered data service are called just once, and are passed the complete list of all the logical hosts that this physical host now masters as arguments to the method call. In Sun Cluster 2.x, the implementation of reconfiguration tends to be that logical hosts are moved one at a time. Thus, each data service will have its methods called multiple times, once for each logical host that is moving.

Working With the Differences Between Solstice HA 1.3 and Sun Cluster 2.x

This section describes some ways in which you can adjust your applications to deal with the differences in the API.

The API definition, and both of its implementations, ultimately require that a method callback be “idempotent,” that is, that it can be called multiple times and that has the same effect as a single call. Pragmatically, a called back method needs to be prepared to deal with the scenario that it has no real work to do, that the work was already accomplished during some previous call to the method. Concretely, this means that the method needs to contain logic that figures out whether there is any work to do for the logical host(s) that are moving. If not, the method should just return. An example of this in shown in Chapter 2 “Sample Data Service”.

These differences in the API implementations should have minimal impact given that a data service’s called-back methods must deal with the basic idempotence issue anyway.

CHAPTER **2**

Sample Data Service

This chapter describes the Sun Cluster Data Services API sample application, `in.named`. The `in.named` daemon is the Solaris implementation of the Internet Domain Name Service (DNS).

Overview

The sample application described in this chapter demonstrates how to make a data service application highly available. *This sample is for illustrative purposes only*. There is no guarantee that this particular application will be highly available.

This chapter assumes that you have read the Sun Cluster Data Services API man pages describing `hareg(1M)` and `haget(1M)`.

This sample application demonstrates many, but not all, of the features included in the API. Note these aspects of the sample application:

- The data used in the sample application cannot be split easily into disjoint sets, so only one data set (one logical host) is used.
- The `in.named` data service supports a command line option `-b`, which points to data in a file system residing on the logical host's disk set.
- In the client-server protocol for a data service, the server sometimes will return its own host name to the client as part of the contents of a message to the client. For such protocols, the client might be depending on this returned host name as the host name to use in the future for contacting the server. For the `in.named` sample, these issues do not arise. The `in.named` data service does not need the host name of the server and does not return the host name to clients.
- The `in.named` data service works off-the-shelf with multihomed hosts.

- The in.named data service works off-the-shelf with the additional IP addresses for the logical hosts. Its stopping and starting methods kill and restart the in.named daemon.

Setting Up the Sample Application

The in.named data service uses only one logical host, even when the underlying cluster has more than one logical host. The method implementations will compute dynamically which logical host is being used. For example, if the hahost1 logical host is used, then the in.named data is placed on the hahost1 disk set.

An administrator can place the boot file (pointed to by the -b flag argument) on any arbitrary file system in the diskset, depending on which file system has space. However, the HA-in.named method implementations need a specific starting point from which to find the boot file. The sample application places this starting point in the administrative file system under the hainnamed subdirectory. It is placed in the hainnamed.config configuration file, which contains a single directory name that indicates a directory elsewhere in the logical host's multihosted disk. This is where the data actually resides (it is a level of indirection).

For our hahost1 logical host, the path name for the file hainnamed.config is /hahost1/hainnamed/hainnamed.config

In general, the path name for an arbitrary logical host would be /*loghost*/hainnamed/hainnamed.config

The HA-in.named methods are written to compute dynamically which logical host is being used for HA-in.named by testing, for the presence or absence of this configuration file, for each logical host.

For example, if file systems A1 through A5 reside on the hahost1 diskset, and the administrator chooses to locate the HA-in.named data in the directory /hahost1/A1/hainnamed, then the hainnamed.config file must contain that directory name.

In the /hahost1/A1/hainnamed directory, the administrator must create a named.boot file for in.named. (See the in.named(1M) man page for information about the contents of the named.boot file.) The administrator updates the in.named database by editing the named.boot file in this directory, just as he or she would edit the /etc/named.boot file in a non-HA in.named configuration. See "Administering HA-in.named: Updating the Database" on page 27, for additional discussion of administration and updates.

Basic Functionality of the in.named Method Implementations

Consider the basic functionality of the HA-in.named method implementations. The start method is not registered in this case, and all the work is accomplished in the start_net method. Similarly, the stop method is not registered for HA-in.named, and all the work is accomplished in the stop_net method. The start_net method starts up the in.named daemon, and the stop_net method kills the in.named daemon by sending a -TERM signal.

The Sun Cluster API requires each method to be idempotent—that is, repeated calls on a method must have the same effect as a single call on that method. For HA-in.named, the idempotency is achieved by having each method test whether its work has already been accomplished. That is, start_net tests whether the in.named daemon is already running, and stop_net tests whether the in.named daemon is already stopped.

The Sun Cluster process monitor facility consists of two components, the pmfadm(1M) command and the rpc.pmfd(1M) process monitor daemon. In the sample application, the pmfadm(1M) command is used to start and kill the in.named daemon, and to query whether the in.named daemon is already running. See the pmfadm(1M) and rpc.pmfd(1M) man pages for details.

The HA-in.named method implementations use the haget(1M) utility program to extract information about the Sun Cluster configuration. (See the haget(1M) man page for details.) The method implementations log their error messages to syslog(3), because the code runs without user attendance. They use the same syslog facility that Sun Cluster uses. Determine the syslog facility name by calling haget(1M) with the option -f syslog_facility.

start_net Method for the in.named Data Service

The following is a sample start_net method for the in.named data service.

```
#! /bin/sh

#

#Copyright 13 Apr 1996 Sun Microsystems, Inc. All Rights Reserved.

#

#ident  "@(#)innamed_start_net.sh  1.1  96/04/13 SMI"

#
```

(continued)

```
# HA-in.named start_net method

ARGV0≐basename $0̀

SYSLOG_FACILITY≐haget -f syslog_facilitỳ
MASTERED_LOGICAL_HOSTS="$1"

if [ -z "$MASTERED_LOGICAL_HOSTS" ]; then

 # This physical host does not currently master any logical hosts.
 exit 0
fi
# Replace comma with space to form an sh word list:
MASTERED_LOGICAL_HOSTS="̀echo $MASTERED_LOGICAL_HOSTS | tr "," "̀"

# Dynamically search the list of logical hosts which this physical

# host currently masters, to see if one of them is the logical host

# that HA-in.named uses.
MYLH=
for LH in $MASTERED_LOGICAL_HOSTS ; do

 # Map logical hostname to administrative file system name:

 PATHPREFIX_FS≐haget -f pathprefix $LH‘
 CONFIG="${PATHPREFIX_FS}/hainnamed/hainnamed.config"

 if [ -f $CONFIG ]; then

   MYLH=$LH

   break

 fi

done

if [ -z "$MYLH" ]; then

 # This host does not currently master the logical host

 # that HA-in.named uses.

 exit 0

fi
```

```
# This host currently masters the logical host that HA-in.named uses, $MYLH
# See if in.named is already running, if so exit. (We must have

# started it on some earlier cluster reconfiguration when this

# physical host first took over mastery of the $MYLH logical host.)
# We determine whether in.named is already running by using the pmfadm
# command to query its status: if the query succeeds, it is already
# running.
if pmfadm -q hainnamed >/dev/null 2>&1 ; then
 exit 0

fi

HA_INNAMED_DIR="̀cat $CONFIG̀"
if [ ! -d $HA_INNAMED_DIR ]; then
 logger -p ${SYSLOG_FACILITY}.err \
   "${ARGV0}: directory $HA_INNAMED_DIR missing or not mounted"

 exit 1

fi

# We cd to the HA_INNAMED_DIR directory because the named.boot file
# contains the names of other files. By cd"ing, we permit all of

# those names to be relative names, relative to the current directory

cd $HA_INNAMED_DIR

if [ ! -s named.boot ]; then
 logger -p ${SYSLOG_FACILITY}.err \

   "${ARGV0}:file $HA_INNAMED_DIR/named.boot is missing or empty"

 exit 1

fi

# Run the in.named daemon under the control of the Sun Cluster process
# monitory facility. Let it crash and restart up to 4 times an hour;
# if it crashes more often than that, the process monitor facility daemon
# will cease trying to restart it.
pmfadm -c hainnamed -n 4 -t 60 /usr/sbin/in.named -b named.boot
if [ $? -ne 0 ]; then
 logger -p ${SYSLOG_FACILITY}.err \
   ‘‘${ARGV0}: pmfadm -c of in.named failed’’
 exit 1
fi

exit 0
```

`stop_net` Method for the `in.named` Data Service

The following is a sample `stop_net` method for the `in.named` data service.

```
#! /bin/sh

# Copyright 13 Apr 1996 Sun Microsystems, Inc. All Rights Reserved.
#

#ident "@(#)innamed_stop_net.sh  1.1 96/04/13 SMI"

#

# HA-in.named stop_net method

#
ARGV0=̀basename $0̀

SYSLOG_FACILITY=̀haget -f syslog_facilitỳ

NOT_MASTERED_LOGICAL_HOSTS="$2"

if [ -z "$NOT_MASTERED_LOGICAL_HOSTS" ]; then

 # This physical host currently masters all logical hosts.

 exit 0

fi

# Replace comma with space to have an sh word list: NOT_MASTERED_LOGICAL_HOSTS=̏echo $NOT_MASTERED_LOGICAL_HOSTS |tr ",
# Dynamically search the list of logical hosts that this physical

# host should not master, to see if one of them is the logical host

# that HA-in.named uses.  There are two cases to consider:

# (1) This physical host gave up mastery of that logical host during

# some earlier cluster reconfiguration.  In that case, the HA administrative

# file system for the logical host will no longer be mounted so the

# /HA administrative_file_system/hainnamed directory will not exist.

# This method has no work to do, because the work got done during the

# earlier cluster reconfiguration when this physical host first gave up

# mastery of the logical host.

# (2) This cluster reconfiguration is the one in which this physical

# host is giving up mastery of the logical host.  In that case, the

# administrative file system is still mounted when the stop_net method

# is called and the /HA administrative_file_system/hainnamed directory

# will exist.
```

```
MYLH=

for LH in $NOT_MASTERED_LOGICAL_HOSTS ; do

 # Map logical hostname to pathprefix file system name:

 PATHPREFIX_FS=`haget -f pathprefix $LH`

 CONFIGDIR="${PATHPREFIX_FS}/hainnamed

 if [ -d $CONFIGDIR ]; then

   MYLH=$LH

   break

 fi

done

if [ -z "$MYLH" ]; then

 # This host is not giving up mastery of the HA-in.named logical host

 # during this cluster reconfiguration.

 exit 0

fi
# This host is giving up mastery of the HA-in.named logical host, $MYLH

# during this cluster reconfiguration.
#
# See if in.named is running, and if so, kill it. If it is not running,

# then either we must have killed it during some earlier reconfiguration

# when this physical host first gave up mastery of the logical host, or

# this physical host has not had mastery of the logical host since it

# last rebooted.

#

# Tell process monitor to kill the in.named daemon, if it was already

# running.
if pmfadm -q hainnamed; then
 pmfadm -s hainnamed TERM
 if [ $? -ne 0 ]; then
   logger ${SYSLOG_FACILITY}.err \
    "${ARGV0}: pmfadm -s of in.named failed"
   exit 1
 fi
```

(continued)

(Continuation)

```
fi

exit 0
```

abort_net Method for the in.named Data Service

The abort method is not registered for the HA-in.named example. The abort_net method uses the same code as the stop_net method; when HA-in.named is registered with Sun Cluster by the hareg(1M) utility, the abort_net registration points to the code used by stop_net.

Setting Timeout Values for in.named Methods

When you register your data service using hareg(1M), you can specify timeout values for methods you have created, such as the start_net, stop_net, and fm_start methods. However, the timeout values you set for your methods must be less than half the timeout value set for logical host takeover during cluster reconfiguration. The default timeout value for logical host takeover is 180 seconds. Therefore, if the timeout values you set for your methods are greater than 90 seconds, you must increase the timeout value for logical host takeover. Otherwise, your methods will time out.

You can increase the logical host takeover timeout values with the scconf(1M) command. Refer to the scconf(1M) man page and to ??? Section 3.15, “Configuring Timeouts for Cluster Transition Steps,” in the *Sun Cluster 2.2 System Administration Guide* for details.

Improving the in.named Methods

Consider some possible improvements to the start_net and stop_net methods for HA-in.named. The methods can benefit from better error detection and handling. For example, you can test whether the /usr/sbin/in.named binary exists, is executable, and is non-empty. If not, an error message can be logged. Before attempting to cat(1) the file hainnamed.config, verify that the file exists, exhibits the correct permissions, and is non-empty.

The methods also can test for the existence of the non-HA in.named data file /etc/named.boot. If the file exists, there is confusion about whether this host is running non-HA in.named or HA-in.named; only one can run at a time. The code can treat this case as a severe configuration error, log appropriate messages, and neither start nor kill in.named.

DNS Clients

In Solaris, a host that is a client of DNS has an /etc/resolv.conf file. The file lists name server hosts to contact for DNS service. The name server hosts are listed as IP addresses rather than host names. More than one host IP address might be listed.

Network clients of HA-in.named would list the IP address of the logical host, for example, that of hahost1, in the /etc/resolv.conf file.

There are periods when a physical host does not master the logical host that HA-in.named uses. However, the host must have the ability to be a client of HA-in.named during those periods. To achieve this, add the IP address of the logical host to the /etc/resolv.conf file on all physical hosts of the cluster.

Administering HA-in.named: Updating the Database

Administration of HA-in.named resembles that of non-HA in.named. To update the in.named database, log in to the server (it is a security risk to grant root NFS access to the file system where the in.named data files are stored). For HA-in.named, log in to the physical server that currently masters the logical host that HA-in.named has been configured to use. Use the hastat(1M) utility to determine which physical host masters which logical hosts.

You perform an update to HA-in.named by editing its data files. Do this in a way that leaves the data files well-formed in the event of a sudden crash. For example, after logging in, cd to the directory where the HA-in.named data is stored (in our example, the directory /hahost1/A1/hainnamed). Then edit a new temporary copy of the data file, and once you are finished, move this copy onto the real data file name. For example:

```
% cd /hahost1/A1/hainnamed
% cp named.boot named.boot.new
% vi named.boot.new
% sync
% mv named.boot.new named.boot
```

As explained in the in.named(1M) man page, you then can use the kill(1M) command to send a SIGHUP signal to the in.named daemon, to cause it to re-read the file.

Documenting HA-in.named

You must document the installation and configuration of the highly available data service. This documentation must explain how to configure any administrative files that live in the administrative file system, and how to install the data service"s data on one or more of the logical host"s file systems or raw partitions. You should also document administration history and updates for the HA version of your data service.

Fault Monitoring Methods for the in.named Data Service

Sun Cluster enables the author of an HA data service to write fault monitoring methods for the data service. As an example, one can write a modest fault monitor for in.named, and can query in.named periodically using nslookup(1M). If the look-up times out using a very long time-out value, the fault monitor will conclude that the in.named daemon is hung and must be killed and restarted.

Fault monitoring will be executed only on the physical host on which in.named is running, that is, on the host that masters the logical host used by in.named. The non-master physical hosts do not perform fault monitoring.

The fault monitor is started by the FM_START method and stopped by the FM_STOP method. It has no need for the FM_INIT method—HA-in.named would not register an FM_INIT method when calling hareg(1M).

The following is a sample FM_START method for the in.named data service.

```
#! /bin/sh

# Copyright 26 Oct 1996 Sun Microsystems, Inc.  All Rights Reserved.

#ident "@(#)innamed_fm_start.sh  1.1  96/04/13 SMI"

# HA in.named fm_start method

# Called-back by Solaris Cluster as the FM_START method for HA in.named.

#
```

(continued)

(Continuation)

```
ARGV0=`basename $0`
SYSLOG_FACILITY=`haget -f syslog_facility`

MASTERED_LOGICAL_HOSTS="$1"
if [ -z "$MASTERED_LOGICAL_HOSTS" ]; then
  # This physical host does not currently master any logical hosts.
  exit 0
fi

# Replace comma with space to form an sh word list:
MASTERED_LOGICAL_HOSTS="`echo $MASTERED_LOGICAL_HOSTS  tr "," " "`"

# Dynamically search the list of logical hosts which this physical
# host currently masters, to see if one of them is the logical host
# that HA-in.named uses.
```

```
MYLH=
for LH in $MASTERED_LOGICAL_HOSTS ; do
 # Map logical hostname to administrative file system name:
 PATHPREFIX_FS=`haget -f pathprefix $LH`
 CONFIG="${PATHPREFIX_FS}/hainnamed/hainnamed.config"

 if [ -f $CONFIG ]; then
   MYLH=$LH
   break
```

(continued)

(Continuation)

```
 fi
done
if [ -z "$MYLH" ]; then
 # This host does not currently master the logical host
 # that HA-in.named uses.
 exit 0
fi

# This host currently masters the logical host that HA in.named uses,
# $MYLH.
# Create an asynchronous process to periodically probe the in.named
# daemon, under the control of the process monitor facility.
# The asynchronous probe is in its own shell script:
#     hainnamed_fmprobe
# The asynchronous process will be terminated by the FM_STOP method.
pmfadm -c hainnamedfm hainnamed_fmprobe $MYLH
exit 0
```

The following is a sample `FM_STOP` method for the `in.named` data service.

```
#! /bin/sh
#
# Copyright 26 Oct 1996 Sun Microsystems, Inc.  All Rights Reserved.
#
#ident "@(#)innamed_fm_stop.sh  1.1  96/04/13 SMI"
#
# HA in.named fm_stop method
#
```

(continued)

(Continuation)

```
# Called back by Sun Cluster as the FM_STOP method for HA in.named.
#
# Stop the asynchronous fault monitoring process that was created
# earlier under the control of pmfd.
#
# Ignore errors when calling pmfadm just in case the hainnamed_fmprobe
# is already not running.  Reasons for it being already not running
# include the fact that it is started only on the physical host that
# currently masters the logical host, the fact that FM_STOP can be
# called even though FM_START has not be en called, and the fact
# that it may have died an early death all by itself.
pmfadm -s hainnamedfm TERM >/dev/null 2>&1
exit 0
```

The following is a sample probe script, `ha.innamed_fmprobe`, for the `in.named` data service. It is started under the control of the process monitor facility by the `FM_START` method.

```
#! /bin/sh
#
# Copyright 26 Oct 1996 Sun Microsystems, Inc.  All Rights Reserved.
#
#ident "@(#)hainnamed_fmprobe.sh  1.1  96/04/13 SMI"
#
# Usage: hainnamed_fmprobe logical_host
#
# Periodically probes the in.named running on the logical_host.
```

(continued)

```
# If the probe times out, then this script will query the pmfd to
# see if the pmfd is still running in.named:
# (i) if so, this script assumes that in.named is hung and
# sends a KILL signal to the in.named process, causing it to
# die.  pmfd will restart in.named provided it has not used
# up its ration of restarts per time period.
# (ii) if not, this script will assume that in.named has exhausted
# its ration of restarts.  This script will call hactl -g to give up
# mastery of the logical host to some other new master physical host.
#
ARGV0=`basename $0`
LOGICAL_HOST="$1"
SYSLOG_FACILITY=`haget -f syslog_facility`
PROBE_INTERVAL_SECS=60
MIN_PROBE_SECS=`hactl -f min_probe_timeout_secs`
PROBE_TIMEOUT_SECS=`expr $MIN_PROBE_SECS + 180`
CLUSTER_KEY=`hactl -f cluster_key`
NSLOOKUP=/usr/sbin/nslookup
if [ ! -x $NSLOOKUP  -o  ! -s $NSLOOKUP ]; then
 logger ${SYSLOG_FACILITY}.err \
  "${ARGV0}: $NSLOOKUP does not exist or is not executable"
 exit 1
fi
```

```
while true; do
 # Call nslookup under a timeout, using hatimerun.
 # The -norecurse option tells in.named not to consult
 # other name service instances on other hosts beyond the
 # one on $LOGICAL_HOST.
 # The -retry=10000 is telling nslookup to take forever
 # retrying: this means that for a hung server, nslookup
 # will never itself giveup, rather, the timeout on hatimerun
 # will expire first.
 hatimerun -t $PROBE_TIMEOUT_SECS \
  $NSLOOKUP -norecurse -retry=10000 $LOGICAL_HOST $LOGICAL_HOST
 if [ $? -ne 99 ]; then
   sleep $PROBE_INTERVAL_SECS
   continue
 fi

 # Here when the timeout occurred.
 logger -p ${SYSLOG_FACILITY}.err \
  "${ARGV0}: nslookup of in.named on $LOGICAL_HOST timed-out"
 if pmfadm -q hainnamed then
   # The in.named process exists.  Kill it on the
   # assumption that it is hung.  Sleep a short time,
   # and if hainnamed still exists in the pmfd, assume
   # that pmfd is restarting it (it has not yet used
   # up its ration of restarts per time interval.)
   logger -p ${SYSLOG_FACILITY}.err \
    "${ARGV0}: KILLing hung in.named"
   pmfadm -k hainnamed KILL
   sleep 30
```

(continued)

(Continuation)

```
   if pmfadm -q hainnamed; then
     continue
   fi
 fi
 # Here when pmfadm -q says that hainnamed no longer
 # exists in pmfd.  Assume that the ration of restarts
 # was exhausted.  Also assume that something is amiss
 # that moving to a new master could improve.
 logger -p ${SYSLOG_FACILITY}.err \
  "${ARGV0}: in.named restarted too many times, not restarting"
 logger -p ${SYSLOG_FACILITY}.err \
  "${ARGV0}: giving up mastery of $LOGICAL_HOST"
 hactl -g -s hainnamed -k $CLUSTER_KEY -l $LOGICAL_HOST
done
```

CHAPTER **3**

Tips for Writing and Testing HA Data Services

This chapter provides tips for writing and testing new highly available data services.

Overview

This chapter describes how to:

- Modify your data service to better suit your particular task
- Test your data service to ensure that it will operate correctly in the HA environment
- Coordinate dependencies between data services

Deciding Which Methods to Use

This section provides some tips about when to use the `start_net`, `stop_net`, and `abort_net` methods versus using the `start`, `stop`, and `abort` methods.

Generally, it is easier to start, stop, or abort the data service using `start_net`, `stop_net`, or `abort_net,` because the logical network addresses are configured to be up at the point where these methods are called.

To start, stop, or abort a data service, you often will have to invoke the data service"s administrative utilities or libraries. Sometimes, the data service has administrative utilities or libraries that use a client-server networking interface to perform the administration. That is, an administrative utility makes a call to the server daemon, so the logical network address might need to be up to use the administrative utility or library.

Consider whether your client software will respond differently depending on whether the network interface or the data service comes online first after a reboot, takeover, or switchover. Use the methods that will ensure adequate retries occur before giving up. For example, if your client implementation does minimal retries when it determines that the data service port is not available, ensure that the data service starts before the network interface is configured. In this case, use the `start` method rather than the `start_net` method.

If you use the `stop` or `abort` method, the data service is still up at the point where the logical network address is configured to be down; it is only after the logical network address is configured down that the `stop` and `abort` methods are invoked.

This creates the invariant that the data service"s TCP or UDP service port, or its RPC program number, always appears to be available to clients on the network—except when the logical host network address also is not responding. This invariant is important only if the client code behaves in a significantly different way when it finds that the TCP or UDP service port, or RPC program number, is not responding, but that the logical host's network address is responding. For example, a client might decide to abandon its retry path early in this scenario. This means that the client code is going down a different code path when it receives an explicit error packet back from the server host saying "ICMP port unreachable" or "Program not registered."

You need in-depth knowledge of the client and the data service's client-server networking protocol to know whether a client implementation depends on this invariant.

Using Keep-Alives

If the client-server communication uses a TCP stream, then both the client and the server should enable the TCP keep-alive mechanism. This is applicable even in the non-HA single server case.

Note - Other connection-oriented protocols might also have keep-alive mechanisms.

On the server side, using TCP keep-alives protects the server from wasting resources for a down (or network partitioned) client. If those resources are not cleaned up (in a server that stays up long enough), eventually the wasted resources will grow without bound as clients crash and reboot.

On the client side, using TCP keep-alives enables the client to be notified when a logical host has failed over or switched over from one physical host to another. That transfer of the logical host breaks the TCP connection. However, unless the client has enabled the keep-alive, it would not necessarily learn of the connection break if the connection happens to be quiescent at the time.

For example, consider the case in which the client is waiting for a response from the server to a long-running request. In this scenario, the client's request message has already arrived at the server and has been acknowledged at the TCP layer, so the client's TCP module has no need to keep retransmitting it. The client application is now blocked, waiting for a response to the request.

Where possible, in addition to using the TCP keep-alive mechanism, the client application also should perform its own periodic keep-alive at its level, because the TCP keep-alive mechanism is not perfect in all possible boundary cases. Using an application-level keep-alive typically requires that the client-server protocol supports a null operation or at least an efficient read-only operation such as a status operation.

Testing HA Data Services

You will want to test your data service implementation thoroughly before putting it into a production environment. This section provides suggestions about how to test your implementation in the HA environment. The test cases are suggestions and are not exhaustive. For testing, you need to have access to a test-bed Sun Cluster configuration, so that your work will not impact production machines.

Test that your HA data service behaves properly in all cases where a logical host is moved between physical hosts. These include system crashes and the use of the `haswitch(1M)` and `scadm(1M) stopnode` commands. Test that client machines continue to get service after these events.

Test the idempotency of the methods. An important way to do this is to configure logical hosts with manual mode ON and repeatedly abort and rejoin one physical host, without ever doing an `haswitch(1M)` of a logical host to it. Let the rejoining host complete cluster reconfiguration before aborting it again. Note that when a rejoining host rejoins the cluster, cluster reconfiguration runs, but no logical host is moved between physical hosts during that reconfiguration.

Another way to test idempotency is to replace each method temporarily with a short shell script that calls the original method twice.

To test that your data service properly implements the `abort` and `abort_net` methods, make one physical host look very sick to Sun Cluster, but without crashing the host outright, so that Sun Cluster will take it out of the system on the "last wishes" path. First, do an ha`switch(1M)` of all logical hosts to that physical host. Then make that host appear to be sick by unplugging all the public network connections to that host. Sun Cluster network fault monitoring will notice the problem and take the physical host out of the cluster, using the aborting "last wishes" path.

Coordinating Dependencies Between Data Services

Sometimes, one client-server data service makes requests upon another client-server data service while fulfilling a request for a client. Informally, a data service A depends on a data service B if, for A to provide its service, B must be providing its service.

Sun Cluster enables having dependent data services by providing the `-d` switch to the `hareg(1M)` program. The dependencies affect the order in which Sun Cluster starts and stops data services. See the `hareg(1M)` man page for details.

Determine whether there are any data service dependencies and whether to supply the appropriate `-d` switches to `hareg(1M)`. Sun Cluster does not check the completeness of the supplied `-d` switches.

Decide whether to use the `-d` switches or to omit them and poll for the availability of the other data service(s) in your HA data service's own code. In some cases, polling is required anyway, because the other data service"s `start` method might be asynchronous—it might start the data service but not wait for the data service to actually be available to clients before returning from the `start` or `start_net` method. Database services typically exhibit this behavior because database recovery time is often lengthy.

Dependent Data Service Using Another Back-End Data Service

Some data services store no data directly themselves, but instead depend upon another back-end data service to store all their data. Such a data service translates all read and update requests into calls on the back-end data service. For example, consider a hypothetical client-server appointment calendar service that keeps all of its data in an SQL database such as Oracle. The appointment calendar service has its own client-server network protocol. For example, it might have defined its protocol using an RPC specification language, such as ONC™ RPC.

In the Sun Cluster environment, you can use Sun Cluster HA for Oracle to make the back-end Oracle database highly available. Then, you can write simple methods for starting and stopping the appointment calendar daemon. You can register the appointment calendar data service with Sun Cluster as one that depends upon another Sun Cluster data service, Sun Cluster HA for Oracle. Specify this dependency using the `-d` option to `hareg(1M)`.

The `start` method for Oracle might initiate only database recovery and might not wait for the recovery to complete. Therefore, our calendar data service daemon, once it has been started, must poll waiting for the Oracle database to become available.

APPENDIX **A**

Using Symbolic Links for Multihosted Data Placement

This appendix describes how to use symbolic links to avoid having to modify data service code.

Occasionally an existing data service has the path names of its data files hard-wired, with no mechanism for overriding the hard-wired path names. To avoid modifying the data service's code, symbolic links sometimes can be used.

For example, suppose the data service names its data file with the hard-wired path name `/etc/mydatafile`. You can change that path from a file to a symbolic link whose value points at a file in one of the logical host's file systems. For example, you can make it be a symbolic link to `/hahost1/A1/`*myservicename*`/mydatafile`.

There is a potential problem with this use of symbolic links. That is, sometimes the data service, or one of its administrative procedures, modifies the data file name as well as its contents. For example, suppose that the data service performs an update by first creating a new temporary file, `/etc/mydatafile.new`. Then it renames the temporary file to have the real file name by using the `rename(2)` system call (or the `mv(1)` program):

```
rename("/etc/mydatafile.new", "/etc/mydatafile");
```

By going through the sequence of creating the temporary file, and then renaming it to the real file, the data service is attempting to ensure that its data file contents are always well-formed.

Unfortunately, the `rename(2)` action destroys the symbolic link. The name `/etc/mydatafile` is now a regular file, and is in the same file system as the `/etc` directory, not in the logical host's dual-ported file system. Because the `/etc` file system is private to each host, the data is not available after a takeover or switchover.

The underlying problem in this situation is that the existing data service is not aware of the symbolic link and was not written with symbolic links considered. To use symbolic links to redirect data access into the logical host's file systems, the data service implementation must behave in a way that does not obliterate the symbolic links. So, symbolic links are not a cure-all for the problem of placing data on the logical host's file systems.

APPENDIX **B**

API Man Pages

This appendix contains a quick reference to the syntax for the commands and functions associated with the Sun Cluster APIs.

The syntax for some of the Sun Cluster Data Services API commands or functions is provided in the following sections.

`hactl(1M)`—Controls operations on high availability data services.

```
/opt/SUNWcluster/bin/hactl [-n] -t|-g -s service_name

-l|-p hostname [-L severity] [-k cluster_key]
hactl [-n] -r -s service_name [-k cluster_key]
hactl -f fieldname
```

`haget(1M` —Queries the current state of cluster configuration.

```
haget [-S] [-a APIversion] -f fieldname [-h hostname] [-s dataservicename]
```

`halockrun(1M)`—Runs a child program while holding a file lock.

```
halockrun [-vsn] [-e exitcode] lockfilename prog [args]
```

`hareg(1M)`—Controls registration and activation of high availability data services.

```
hareg -r service_name -m method=path[, method=path] ... [-b basedir]
 [-t method=timeout[, method=timeout] ...]     [-d depends_on_service[, ...]] [-h logical_host [, ...]]

 [-v service_version] [-a APIversion] [-p pkg[, ...]]

hareg -s -r Sun_service_name [-h logical_host [, ...]

hareg -u service_name

hareg -q service_name [-M method | -T method | -D | -V | -A
 | -P | -B]

hareg -y|-n service_name[, ...]

hareg [-Y | -N]
```

`hatimerun(1M)`—Runs a child program under a time-out.

```
hatimerun [-va] [-k signalname] [-e exitcode]

 -t timeOutSecs prog args
```

`pmfadm(1M)`—Processes monitor facility administration.

```
pmfadm -c nametag [-n retries] [-t period] [-a action]

 command [args_to_command ...]
pmfadm -m nametag [-n retries] [-t period]
pmfadm -s nametag [-w timeout] [signal]
pmfadm -k nametag [-w timeout] [signal]
pmfadm -l nametag [-h host]
pmfadm -q nametag [-h host]
```

`ha_get_calls(3HA)`—Gets high availability environment. Includes `ha_get_calls`, `ha_getconfig`, `ha_getcurstate`, `ha_getmastered`, `ha_getnotmastered`, `ha_getonoff`, `ha_getlogfacility`.

```
cc [flag...] -I/opt/SUNWcluster/include file...
 -L /opt/SUNWcluster/lib [threads lib] -lhads -lintl -ldl
 -lnsl -lccd -lclustm -lcdb -lcluster [library...]

ha_error_t ha_getconfig(ha_handle_t handle, ha_config_t
 **config);

ha_error_t ha_getcurstate(ha_handle_t handle, ha_lhost_dyn_t
 **lhosts[]);

ha_error_t ha_getmastered(ha_handle_t handle, ha_lhost_dyn_t
 **lhosts[]);

ha_error_t ha_getnotmastered(ha_handle_t handle, ha_lhost_dyn_t
 **lhosts[]);

ha_error_t ha_getonoff(ha_handle_t handle, char *service_name,
 boolean_t *ison);

ha_error_t ha_getlogfacility(ha_handle_t handle, int *facility);
```

ha_open(3HA) / ha_close(3HA)—High availability environment open/close.

```
cc [flag...] -I/opt/SUNWcluster/include file...
 -L /opt/SUNWcluster/lib [threads lib] -lhads -lintl -ldl
 [library...]

ha_error_t ha_open(ha_handle_t *handlep);

ha_error_t ha_close(ha_handle_t handle);
```

hads(3HA)—Library routines for high availability data services.

```
cc [flag...] -I/opt/SUNWcluster/include file...
 -L /opt/SUNWcluster/lib [threads lib] -lhads -lccd -lcdb -lcluster -lclustm
 [library...]

#include <hads.h>
```