Solstice HA 1.3 Programmer's Guide

THE NETWORK IS THE COMPUTER™

SunSoft, Inc.
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043 USA
415 960-1300 fax 415 969-9131

Part No.: 805-0318-10
Revision A, April 1997

Please
Recycle

Adobe PostScript

Contents

Preface

Ultra™ Enterprise™ Cluster HA is a hardware and software product that supports specific dual-server hardware configurations. It is compatible with the Solaris™ 2.5.1 software environment. When configured properly, the hardware and software together provide highly available data services. The software component, Solstice™ High Availability (Solstice HA), depends upon the mirroring and diskset capabilities and other functionality provided by Solstice DiskSuite™ 4.1, which is an integral part of Solstice HA.

Solstice HA provides two application programming interfaces (API) for making data services highly available. The data services API permits client-server data services to be layered on top of Solstice HA. The fault monitor API enables programmers to develop fault monitors for a new highly available 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 Solstice HA environment. This guide provides tips on how to achieve the addition.

The *Solstice HA 1.3 Programmer's Guide* describes the recommended usage of the two APIs. It also discusses conventions that a data service should follow to be highly available.

Part 1 – Data Services API includes an example of 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 book is intended to be used with the other hardware and software books listed under "Related Documentation" on page vii, and with the man pages associated with the API. Some of 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)`, `halockrun (1m)`. Of particular interest is also the *Solstice HA 1.3 User's Guide*. This book describe the HA environment into which you will integrate your data service.

Who Should Use This Book

This book is 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 he or she is integrating.

How This Book Is Organized

This document contains the following chapters and appendixes:

Part 1 – Data Services API

Chapter 1, "Data Services API Introduction," introduces the Solstice HA concepts that enable application programs to become highly available.

Chapter 2, "Sample Data Service," describes a sample data service used to demonstrate how the API is used.

Chapter 3, "Tips for Writing and Testing HA Data Services," provides suggestions for how to most effectively write and test new data services.

Appendix A, "Using Symbolic Links for Dual-Ported Data Placement," describes how you can use symbolic links to avoid having to modify data service code.

Appendix B, “API Man Pages,” contains quick reference to the syntax for the commands and functions associated with the Solstice HA Data Services API, and the complete text of the man pages.

Related Documentation

The documents in Table P-1 contain information that may be helpful to the system administrator and service provider.

Table P-1 Related Documentation

Product Family	Title	Part Number
Solstice HA	*Solstice HA 1.3 User’s Guide*	805-0317
	Solstice HA 1.3 Software New Product Information	805-0629
Name Services	*Name Services Administration Guide*	801-6633
	Name Services Configuration Guide	801-6635
Other Referenced Manuals	*NFS Administration Guide*	801-6634
	TCP/IP Network Administration Guide	801-6632

Typographic Conventions

Table P-2 describes the typographic conventions used in this book.

Table P-2 Typographic Conventions

Typeface or Symbol	Meaning	Example
`Typewriter`	The names of commands, files, and directories; on-screen computer output.	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% You have mail.`
`boldface`	What you type, contrasted with on-screen computer output.	`machine_name% su` `Password:`
italic	Command-line placeholder: replace with a real name or value. Book titles, new words or terms, or words to be emphasized.	To delete a file, type `rm` *filename*.

Shell Prompts in Command Examples

Table P-3 shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

Table P-3 Shell Prompts

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

Data Services API Introduction 1

This chapter introduces the Solstice HA Data Services API and the concepts needed to make your data service applications highly available.

1.1 Overview

The Solstice HA 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 allows 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)` - control registration and activation of Solstice HA data services
- `haget(1M)` - query current state of Solstice HA configuration
- `hads(3ha)` - library routines for Solstice HA data services
- `ha_open(3ha)`, `ha_close(3ha)`- Solstice HA environment open and close
- `ha_get_calls(3ha)` - get Solstice HA environment

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

1.2 Interaction Between Data Services and Solstice HA

This section introduces the interface between a data service and Solstice HA.

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

After the failure of a host, Solstice HA itself takes care of moving the logical host (both its diskset and its logical network IP addresses) to the surviving host. At this point, the data service's software must be restarted on the surviving host. Solstice HA 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` method.

The Solstice HA `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 Solstice HA to coordinate this shut-down work with layered data services, each data service also registered a `stop` method. Solstice HA calls the data service's `stop` method during `haswitch(1M)` operations, and whenever Solstice HA is stopped using `hastop(1M)`. The `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.

Solstice HA continuously monitors the health of the physical servers in the cluster. In some cases, Solstice HA will decide that a physical server is failing, but is still able to execute some “last wishes” cleanup code before Solstice HA halts and reboots the server. In this case, each data service is given an opportunity to execute last wishes cleanup code before Solstice HA halts the server. Solstice HA does this by calling the `abort` 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.

1.3 Logical Host Configuration Issues

A data service is made highly available by exploiting the Solstice HA 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.

1.3.1 Data Service Use of Single or Multiple Logical Hosts

In release 1.3 of Solstice HA, there are at most two logical hosts. This might change with later releases, so your data service implementation should not depend on this fact. You must decide whether your data service will keep its data in just one or in multiple logical hosts.

It is generally easier to design and implement a data service that uses just one logical host. In that case, all the data service's data is placed on just that logical host's diskset. The data service needs just one set of daemon(s). A physical host runs the daemon(s) 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 daemon(s). When the physical host is giving up mastery of the logical host, the data service's `stop` method can stop the daemon(s). In some cases, killing the daemon(s) 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 Solstice HA-NFS product, which has multiple file systems with different data residing in each file system. For Solstice 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 masters 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 – Have the data service use just one of the logical hosts, unless the data is easily split into disjoint collections *and* the rudimentary load balancing that using multiple logical hosts enables is a significant benefit.

1.3.2 Required File System for Each Logical Host

Each Solstice HA logical host has a diskset. The diskset might contain one or more file systems or raw partitions. Solstice HA 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 Solstice HA is first installed and configured, the `hasetup(1M)` program assists the administrator in creating the required file system, thus following the required convention. Solstice HA uses the term *HA administrative file system* to refer to this special required file system.

For example, suppose there is a logical host named “hahost1” and also a diskset “hahost1.” On the hahost1 diskset, the HA administrative file system is mounted in the name space as “/hahost1.”

Data service code should not assume that the HA administrative file system always has a name of the form “/logicalhostname,” instead, it should use the API calls that take the logical host name as an argument and return the name of the HA administrative file system (the `pathprefix` field described in `haget(1M)`). See `haget(1M)` and `hads(3HA)` for more information.

The HA administrative file system is used by Solstice HA for some of its own administrative data. Layered data services can also use it for some of their administrative data. Data services should use the HA administrative file system rather than private file systems on the physical hosts to avoid having to keep multiple copies (one copy on each physical host) in agreement. Solstice HA does not provide any interfaces for keeping multiple copies of data in agreement. This problem does not arise when the data is stored on the logical host, since the data is accessible when the physical host masters that logical host.

1.3.2.1 Required HA Administrative File System Conventions

If your data service uses the HA 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 HA administrative file system. For example, if the data service uses Solaris packages, then the subdirectory should have a name of the form:

/HA_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 Solstice HA 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 HA administrative file system is relatively small. Each data service should limit the amount of administrative data it keeps in the HA administrative file system to a few kilobytes. If larger administrative data is required, use the HA administrative file system as a level of indirection—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 HA administrative file system, because for most data services, that data would be too large.

1.4 Data Service Requirements

This section presents the requirements that a data service must meet to participate in the Solstice HA Data Service API.

1.4.1 Client-Server Environment

Solstice HA is designed for client-server networking environments. Solstice HA 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.

1.4.2 Crash Tolerance

The data service must be crash-tolerant. This means that the data service's daemon process(es) 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, Solstice HA 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.

1.4.3 Dual-Ported Data

The logical host's diskset is dual-ported, so that when one physical host crashes, the surviving host 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 has hard-wired file 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 having to change the data service code. See Appendix A, “Using Symbolic Links for Dual-Ported Data Placement” for a more detailed discussion about using symbolic links.

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

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

1.4.4 Host Names

You must determine whether the data service ever needs to have the host name of the server on which it is running. If so, then 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 Solstice HA 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/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.

1.4.5 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. Solstice HA 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 Solstice HA 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 Solstice HA logical host, the set of host name/IP address pairs is part of the Solstice HA configuration data and is specified by the system administrator when Solstice HA is first installed and configured. The Solstice HA 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. The semantics of `INADDR_ANY` are that it effectively binds to all the 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 Solstice HA logical host's IP addresses.

1.4.6 Binding to `INADDR_ANY` Versus Binding to Specific IP Addresses

Even in the non-multihomed case, the Solstice HA logical host concept has the effect that the machine can 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 Solstice HA 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 daemon(s).

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 Solstice HA. 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 individually binding to individual IP addresses, this becomes a non-issue. If the data service's stopping and starting methods accomplish their work by killing and restarting the data service's daemon(s), then the data service stops and starts using the network addresses at the appropriate times.

1.4.7 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.

1.5 Registering a Data Service

A data service is registered with Solstice HA using the Solstice HA program `hareg(1M)`. Registration is persistent in that it survives across takeovers, switchovers, and reboots. Registration with Solstice HA 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 additional details.

In addition to the distinction between registered versus unregistered, Solstice HA also 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 having to take the more drastic step of unregistering it.

For example, a system administrator could put the data service in the “off” state 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 Solstice HA 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 Solstice HA, 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 Solstice HA cluster reconfiguration.

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

Sample Data Service 2

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

2.1 Overview

The sample application described in this chapter demonstrates how to make a data service application highly available. It 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 Solstice HA 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 has a command line option, `-b`, used to point to a data file. The `-b` option points to data in a file system residing on the logical host's diskset.

- 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.

2.2 Sample Application Setup

The `in.named` data service uses only one logical host, even when the underlying Solstice HA 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” diskset.

An administrator may 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 HA administrative file system under the `hainnamed` subdirectory. It is placed in the configuration file `hainnamed.config`, which contains a single directory name that indicates a directory elsewhere in the logical host's dual-ported 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:

/HA administrative_file_system`/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 "HA-in.named Administration: Updating the Database" on page 2-10 for additional discussion of administration and updates.

2.2.1 *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 it by sending it a `-TERM` signal.

The Solstice HA 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 Solstice HA 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 Solstice HA 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 Solstice HA uses. Determine the syslog facility name by calling `haget(1M)` with the option `-f syslog_facility`.

2.2.2 *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"
#
# HA-in.named start_net method
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 HA 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
```

continued:

```
# 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 $CONFIG`"
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 Solstice HA 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
```

2.2.3 *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 $0`
SYSLOG_FACILITY=`haget -f syslog_facility`
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
# HA administrative file system is still mounted when the stop_net method
# is called and the /HA administrative_file_system/hainnamed directory
# will exist.
```

continued:

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

exit 0
```

2.2.4 *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 Solstice HA by the `hareg(1M)` utility, the `abort_net` registration points to the code used by `stop_net`.

2.2.5 *Possible Improvements to* `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, has 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 one time. The code can treat this case as a severe configuration error, log appropriate messages, and neither start nor kill `in.named`.

2.2.6 *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 Solstice HA cluster.

2.2.7 HA-in.named Administration: 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.

2.2.8 HA-in.named Documentation

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 HA 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.

2.3 *Fault Monitoring Methods for the* `in.named` *Data Service*

Solstice HA 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 Solstice HA as the FM_START method for HA in.named.
#
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.
```

continued

continued:

```
MYLH=
for LH in $MASTERED_LOGICAL_HOSTS ; do
      # Map logical hostname to HA 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.
# 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
#
# Called back by Solstice HA 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 been 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.
# 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 giveup
# 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
```

continued

continued:

```
      # 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
            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
```

Tips for Writing and Testing HA Data Services

3

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

3.1 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

3.2 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 on-line 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 appear 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.

3.3 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 a keep-alive mechanism.

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 allows 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.

3.4 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 Solstice HA 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 `haswitch(1M)` and `hastop(1M)`. Test that client machines continue to get service after these events. Try crashing either host in the middle of `haswitch(1M)` and `hastop(1M)`. For example, try crashing the host:

- to which a logical host is moving.
- from which the logical host is moving.
- that is executing the `hastop(1M)` command.
- that is not executing the `hastop(1M)` command.

Test halting both physical hosts and then starting them up again at approximately the same time.

Test the idempotency of the methods. An important way to do this is to repeatedly crash and reboot one physical host, without ever doing an `haswitch(1M)` of a logical host to it. Let the rebooting host complete cluster reconfiguration before crashing it again. Note that when a rebooting 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 temporarily each method 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 Solstice HA, but without crashing the host outright, so that Solstice HA will take it out of the system on the “last wishes” path. First, do an `haswitch(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. Solstice HA's network fault monitoring will notice the problem and take the physical host out of the cluster, using the aborting “last wishes” path.

3.5 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.

Solstice HA enables having dependent data services by providing the `-d` switch to the `hareg(1M)` program. The dependencies affect the order in which Solstice HA 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)`. Solstice HA 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.

3.5.1 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 Solstice HA environment, you can use Solstice HA-DBMS for ORACLE7 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 Solstice HA as one that depends upon another Solstice HA data service, Solstice HA-DBMS for ORACLE7. 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.

Using Symbolic Links for Dual-Ported Data Placement

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

In Chapter 1, “Data Services API Introduction,” we mentioned that 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 could 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 system(s), 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 system(s).

API Man Pages

This appendix contains a quick reference to the syntax for the commands and functions associated with the Solstice HA APIs, and the complete text of the man pages.

The man pages described in this appendix are included in the printed version of this book. These pages are not available in the AnswerBook on-line documentation. They are available on line using the `man(1)` command.

B.1 API Man Pages Quick Reference

The syntax for each Solstice HA Data Services API command or function is included below.

- `hactl(1M)` – Control operations on Solstice HA.

```
/opt/SUNWhadf/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)` – query current state of Solstice HA configuration.

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

- `halockrun(1M)` – Run a child program while holding a file lock.

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

- `hareg(1M)` – control registration and activation of Solstice HA data services.

```
hareg -r service_name -m method=path[,method=path]... [-b basedir]
    [-t method=timeout[,method=timeout]...]
    [-d depends_on_service[,...]] [-v service_version] [-a APIversion]
    [-p pkg[,...]]

hareg -s -r Sun_service_name

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)` – Run a child program under a time-out.

```
hatimerun [ -va ] [ -k signalname ] [ -e exitcode ]
    -t timeOutSecs prog args
```

- `pmfadm(1M)` – Process 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 ]
```

- `hads(3HA)` – library routines for Solstice HA data services.

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

#include <hads.h>
```

- `ha_get_calls(3HA)` – `ha_get_calls`, `ha_getconfig`, `ha_getcurstate`, `ha_getmastered`, `ha_getnotmastered`, `ha_getonoff`, `ha_getlogfacility` - get Solstice HA environment.

```
cc [flag...] -I/opt/SUNWhadf/include file...
    -L /opt/SUNWhadf/lib [threads lib] -lhads -lintl -ldl
    [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,
    bool_t *ison);

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

- `ha_open(3HA) / ha_close(3HA)` – Solstice HA environment open / close.

```
cc [flag...] -I/opt/SUNWhadf/include file...
    -L /opt/SUNWhadf/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);
```

Index