

Netra™ High Availability Suite 3.0 1/08 Foundation Services SA Forum Programming Guide

Sun Microsystems, Inc.
www.sun.com

Part No. 819-5246-13
March 2008, Revision A

Submit comments about this document at: <http://www.sun.com/hwdocs/feedback>

Sun Microsystems, Inc. has intellectual property rights relating to technology that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents>, and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this 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.

Sun, Sun Microsystems, the Sun logo, Netra, Java, 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.

U.S. Government Rights—Commercial use. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

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 2008 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, Californie 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à <http://www.sun.com/patents> et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats-Unis et dans les autres pays.

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.

Sun, Sun Microsystems, le logo Sun, Netra, Java, docs.sun.com, et Solaris sont des marques de fabrique ou des marques déposées 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 licenciées de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.



Contents

Preface ix

1. Foundation Services SA Forum CLM API 1

Overview 1

Characteristics of the CLM API 2

2. Setting Up the Development Environment 5

Introduction to the Development Environment 5

Setting Up the Development Host 6

Setting Up a Foundation Services Cluster 6

Installing Libraries and Header Files 7

▼ To Install the Developer Package and Trace Package 7

3. Building CLM Applications 9

Installing Applications on a Cluster 9

Setting Up a Makefile 10

Compiling Applications 10

Including Applications in a Startup Script 11

Running Your Applications on the Cluster 11

Application Examples 12

Library Life Cycle 15

4. Retrieving Node Information Using the CLM API	17
Retrieving Information About a Node	17
Retrieving Information About All Nodes in the Cluster	20
Using the <code>saClmClusterNodeT</code> Structure for Information About Cluster Nodes	24
5. Understanding Change Notifications	25
Introduction to Change Notifications	25
Understanding the Structure of Notifications	25
6. Debugging Applications in the Foundation Services	29
Reporting Application Errors	29
Reading Error Information for Debugging	30
Stopping the Daemon Monitor for Debugging	30
Broken Pipe Error Messages	31
Index	33

Figures

- FIGURE 1-1** Interaction of the Cluster Membership Service With Your Applications Using the CLM API 2
- FIGURE 2-1** Setting Up the Development Environment 6

Code Examples

- CODE EXAMPLE 3-1 Mandatory Code Fragment: `common.c` 12
- CODE EXAMPLE 3-2 The `common.h` Header File 12
- CODE EXAMPLE 3-3 Example `NodeGet.c` Program 13
- CODE EXAMPLE 3-4 Makefile for the `NodeGet.c` Program 14
- CODE EXAMPLE 4-1 Retrieving Information About the Local Node Using the
`saClmClusterNodeGetAsync()` Function 18
- CODE EXAMPLE 4-2 Synchronous `saClmClusterTrack()` Function 20
- CODE EXAMPLE 4-3 The `print_notif_buffer()` function 21
- CODE EXAMPLE 4-4 The `saClmClusterTrackAsync()` Function 22
- CODE EXAMPLE 5-1 Application With All Notifications Provided 27

Preface

This book describes how to write applications that use the Service Availability Forum (SA Forum) Cluster Membership (CLM) API for the Netra™ High Availability (HA) Suite Foundation Services 3.0.

Who Should Use This Book

This book is for application developers who are writing programs for clusters running the Foundation Services. This book describes how to perform the following tasks:

- Create and use a development environment
 - Develop, compile, link, and execute applications across the cluster
 - Monitor and manage nodes in the cluster using the SA Forum CLM API
-

Before You Read This Book

To write applications for the Foundation Services, you must have experience with the C programming language. Knowledge of using and deploying highly available applications on a cluster, and knowledge of the developer tools offered by the Solaris™ Operating System (Solaris OS) is an advantage.

Before reading this book, read the *Netra High Availability Suite 3.0 1/08 Foundation Services Overview*.

How This Book Is Organized

This book is divided into the following chapters:

- [Chapter 1](#) provides an overview of the basic functions and characteristics of the SA Forum CLM API.
- [Chapter 2](#) describes the requirements of a development host on which to write applications that use the SA Forum CLM API.
- [Chapter 3](#) describes how to install, compile, and run your applications.
- [Chapter 4](#) describes how to identify and retrieve data about nodes in the cluster.
- [Chapter 5](#) describes the `nhcmmd` daemon, the notifications this daemon sends, and how to interpret these notifications.
- [Chapter 6](#) describes how to debug your applications. This chapter also describes the return values provided by the SA Forum CLM API.

Shell Prompts

Shell	Prompt
C shell	<i>machine-name%</i>
C shell superuser	<i>machine-name#</i>
Bourne shell and Korn shell	\$
Bourne shell and Korn shell superuser	#

Typographic Conventions

Typeface*	Meaning	Examples
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. % You have mail.
AaBbCc123	What you type, when contrasted with on-screen computer output	% su Password:
<i>AaBbCc123</i>	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. To delete a file, type <code>rm filename</code> .

* The settings on your browser might differ from these settings.

Related Documentation

The following table lists the documentation for this product. The online documentation is available at:

<http://docs.sun.com/app/docs/prod/netra.ha30>

Application	Title	Part Number
Late-breaking news	<i>Netra High Availability Suite 3.0 1/08 Release Notes</i>	819-5249-14
Introduction to concepts	<i>Netra High Availability Suite 3.0 1/08 Foundation Services Overview</i>	819-5240-13
Basic setup, supported hardware, and configurations	<i>Netra High Availability Suite 3.0 1/08 Foundation Services Getting Started Guide</i>	819-5241-13
Automated installation methods	<i>Netra High Availability Suite 3.0 1/08 Foundation Services Installation Guide</i>	819-5242-13
Detailed installation methods	<i>Netra High Availability Suite 3.0 1/08 Foundation Services Manual Installation Guide for the Solaris OS</i>	819-5237-13
Cluster administration	<i>Netra High Availability Suite 3.0 1/08 Foundation Services Cluster Administration Guide</i>	819-5235-13

Application	Title	Part Number
Using the Cluster Membership Manager	<i>Netra High Availability Suite 3.0 1/08 Foundation Services CMM Programming Guide</i>	819-5236-13
Using the SAF CMM API	<i>Netra High Availability Suite 3.0 1/08 Foundation Services SA Forum Programming Guide</i>	819-5246-13
Using the Node Management Agent	<i>Netra High Availability Suite 3.0 1/08 Foundation Services NMA Programming Guide</i>	819-5239-13
Configuring outside the cluster using CGTP	<i>Netra High Availability Suite 3.0 1/08 Foundation Services Standalone CGTP Guide</i>	819-5247-13
Man pages for Foundation Services features and APIs using the Solaris OS	<i>Netra High Availability Suite 3.0 1/08 Foundation Services Solaris Reference Manual</i>	819-5244-13
Man pages for Foundation Services features and APIs using Linux	<i>Netra High Availability Suite 3.0 1/08 Foundation Services Linux Reference Manual</i>	819-5245-12
Definitions and acronyms	<i>Netra High Availability Suite 3.0 1/08 Foundation Services Glossary</i>	819-5238-13
Common problems	<i>Netra High Availability Suite 3.0 1/08 Foundation Services Troubleshooting Guide</i>	819-5248-13

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (<http://www.sun.com/documentation>)
- Support (<http://www.sun.com/support>)
- Training (<http://www.sun.com/training>)

Third-Party Web Sites

Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments by going to:

<http://www.sun.com/hwdocs/feedback>

Please include the title and part number of your document with your feedback:

Netra™ High Availability Suite 3.0 1/08 Foundation Services SA Forum Programming Guide, part number 819-5246-13.

Foundation Services SA Forum CLM API

The Service Availability Forum (SA Forum) Cluster Membership (CLM) API can be used to develop programs for highly available clusters running the Foundation Services. For more information, see the following topics:

- “Overview” on page 1
- “Characteristics of the CLM API” on page 2

Overview

Foundation Services 3.0 is supported for use with the SA Forum CLM API B.02.01, in addition to the existing CMM API.

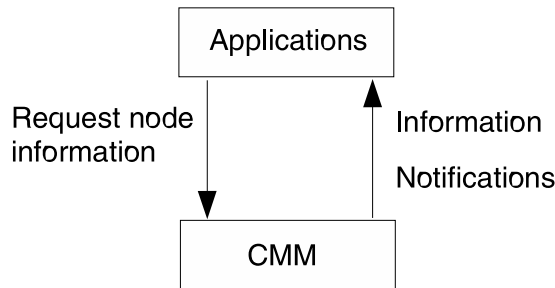
The CLM API provides a programming interface to the Cluster Membership Service, which provides membership information about the nodes in a cluster. This API enables you to write applications that monitor the state of peer nodes. You can use the CLM API to obtain information about which node is a member of the cluster.

More information about the SA Forum/CLM API can be found at:

<http://www.saforum.org/>

This interaction of the CLM API with the Foundation Services is illustrated in [FIGURE 1-1](#).

FIGURE 1-1 Interaction of the Cluster Membership Service With Your Applications Using the CLM API



Characteristics of the CLM API

The CLM API enables you to create highly available applications to determine which nodes are in the cluster. The main functions of the CLM API are as follows:

- Determine the availability of peer nodes
- Gather information about some or all peer nodes

The CLM API has the following multithreading characteristics:

- It is multithread safe: mutual exclusion among threads is guaranteed when critical sections of the API are executed.
- It is deferred-thread cancellation safe: All API functions are cancellation points.
- It is not asynchronous-thread cancellation safe.
- API calls cannot be interrupted by signals: If a signal is caught during a call, the call runs to completion. The call is not aborted and does not return an error in this case.
- It is not signal-handler safe: applications must not make calls to the API from signal handlers.
- It is not `fork1` safe: Applications must not make calls to the API from fork handlers.

The locations of the CLM API header file and the default location for the CLM API library files differ depending on the operating system (OS) used.

- For the Solaris OS, the CLM API header file, `saClm.h`, is located in the `/opt/SUNWcgha/include/saf` directory. The default location of the CLM API library files is the `/opt/SUNWcgha/lib` directory.

- For the Linux OS, the `cmm.h` file, is located in the `/opt/sun/include` directory and the default location of the CMM API library files is the `/opt/sun/lib` directory for 32-bit applications or the `/opt/sun/lib64` directory for 64-bit applications.

To access the CLM API header file and libraries for the Solaris OS, the following packages must be installed in your development environment:

- `SUNWnhas-safclm-headers`
- `SUNWnhas-common-libs`
- `SUNWnhas-cmm-libs`
- `SUNWnhas-safclm-libs`

To access the CLM API header file and libraries for Linux, the following packages must be installed in your development environment:

- `sun-nhas-safclm-headers-3.0-*.x86_amd64.rpm`
- `sun-nhas-common-libs-3.0-*.x86_amd64.rpm`
- `sun-nhas-cmm-libs-3.0-*.x86_amd64.rpm`
- `sun-nhas-safclm-libs-3.0-*.x86_amd64.rpm`

In addition, examples of using SA Forum/CLM will be provided in the `SUNWnhas-safclm-headers` package in the `/opt/SUNWcgha/examples/SaClm_API` directory (Solaris OS) or the `/opt/sun/nhas/examples/SaClm_API` directory (Linux OS).

Setting Up the Development Environment

To develop applications for the Foundation Services, you must set up a development environment for your development host. For information, see the following topics:

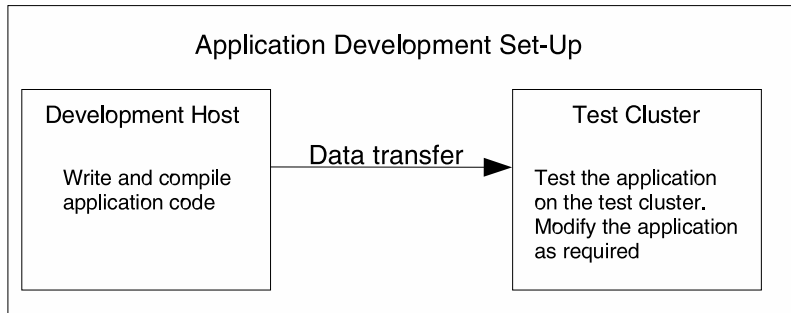
- [“Introduction to the Development Environment” on page 5](#)
- [“Setting Up a Foundation Services Cluster” on page 6](#)
- [“Installing Libraries and Header Files” on page 7](#)

For general information about developing applications on the Solaris Operating System, see the *Solaris Software Developer Collection*.

Introduction to the Development Environment

The development environment for the Foundation Services consists of a development host connected through an installation server to a cluster of nodes. The Foundation Services runs on the nodes of a cluster and does not need to be installed on the development host. Install only the developer packages of the Foundation Services on the development host. This development host together with a cluster for testing your applications is called the development environment. The following figure illustrates the development environment:

FIGURE 2-1 Setting Up the Development Environment



The development environment refers to the set-up with which you work during the development phase, when writing and testing applications.

Setting Up the Development Host

The development host must have at least 1 GBit of disk space, with a minimum of 512 MBytes RAM.

Refer to the *Netra High Availability Suite 3.0 1/08 Release Notes* for the latest information about supported operating systems and software versions.

Setting Up a Foundation Services Cluster

To run applications, you must build them on your development host and deploy them on a cluster that runs the Foundation Services. For information about how to set up a build server and on how to set up a cluster that runs the Foundation Services, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Getting Started Guide* or the Netra HA Suite installation guides.

To install, compile, and run your applications, see [“Installing Applications on a Cluster” on page 9](#).

Installing Libraries and Header Files

Ensure that the library contents and header files of the Foundation Services are available in the runtime environment of your development host. This includes the CLM header (.h) files and library (.so) files. These files are delivered as follows:

- On the Solaris OS, these files are in the `SUNWnhas-safclm-libs` and `SUNWnhas-safclm-headers` developer packages, which can be installed on your development host without having to install a running Foundation Services CLM. The `SUNWnhas-safclm-libs` and `SUNWnhas-safclm-headers` developer packages require that the `SUNWnhas-common-libs` and `SUNWnhas-cmm-libs` packages are present.
- On Linux, these files are in the `sun-nhas-safclm-headers-3.0-*.x86_amd64.rpm` and `sun-nhas-safclm-libs-3.0-*.x86_amd64.rpm` developer packages, which can be installed on your development host without having to install a running Foundation Services CMM. These developer packages require that the `sun-nhas-common-libs-3.0-*.x86_amd64.rpm` and `sun-nhas-cmm-libs-3.0-*.x86_amd64.rpm` packages are present.

You can write your applications on your development host using the CLM API libraries and header files, but you cannot successfully run your applications on your development host. For highly available cluster-based applications, run your applications on a cluster that is running the Foundation Services.

▼ To Install the Developer Package and Trace Package

1. Log into the development host as superuser.

2. Install the packages as follows:

On the Solaris OS:

```
# pkgadd -d /software-distribution-dir/Packages/  
SUNWnhas-cmm-libs SUNWnhas-cmm-headers SUNWnhas-common-libs
```

Where *software-distribution-dir* is the location of the software distribution.

On Linux:

```
# cd /software-distribution-dir/Packages/  
# rpm -i sun-nhas-safclm-headers-3.0-*.x86_amd64.rpm  
sun-nhas-safclm-libs-3.0-*.x86_amd64.rpm  
sun-nhas-common-libs-3.0-*.x86_amd64.rpm  
sun-nhas-cmm-libs-3.0-*.x86_amd64.rpm
```

Where *software-distribution-dir* is the location of the software distribution.

3. **Verify that the `libSaClm.so` library is available in the `/opt/SUNWcggha/lib/` directory (Solaris OS), the `/opt/sun/lib` directory (32-bit applications on Linux), or the `/opt/sun/lib64` directory (64-bit applications on Linux) and that the header files are available in the `/opt/SUNWcggha/include/` directory (Solaris OS) or the `/opt/sun/include` directory (Linux OS).**

Note – The code examples provided in this guide require you to install the library and header files in these default locations.

The CLM API is provided by the `libSaClm.so` library. The `libSaClm` library communicates with the `nhcmmd` daemon, which is monitored by the Daemon Monitor, `nhpmd`. For further information on the `nhcmmd` daemon, see the `nhcmmd1M` (Solaris) or `nhcmmd8` (Linux) man page. For more information on the Daemon Monitor, see the `nhpmd1M` (Solaris) or `nhpmd8` (Linux) man page.

4. **To link the `libSaClm` library to your application, add the `/opt/SUNWcggha/lib/` directory (Solaris OS), the `/opt/sun/lib` directory (32-bit applications on Linux), or the `/opt/sun/lib64` directory (64-bit applications on Linux) to the environment variable `LD_LIBRARY_PATH`.**

The services that these libraries use are only available when you run your applications on the fully installed cluster running the Foundation Services. For more information see [“Setting Up a Foundation Services Cluster” on page 6](#).

Building CLM Applications

For information about how to build applications that use the CLM API, see the following sections:

- “Installing Applications on a Cluster” on page 9
- “Setting Up a Makefile” on page 10
- “Compiling Applications” on page 10
- “Including Applications in a Startup Script” on page 11
- “Running Your Applications on the Cluster” on page 11
- “Application Examples” on page 12

Installing Applications on a Cluster

If your application is to be run on a master-eligible or dataless node, install the application binary files on the node.

If your application is to be run on a diskless node, install the binaries on the master node in:

```
/export/root/diskless_node_name/path
```

where *diskless_node_name* is the name of the diskless node and *path* is the path to the application. For example, if the binaries are installed in the `/opt/mySvc/bin/myapp` directory, the application binaries are installed in the `/export/root/NetraDiskless1/opt/mySvc/bin/myapp` directory for the *NetraDiskless1* diskless node.

If your application will run on a dataless node, binaries can be installed on the local disk (as in the master-eligible node case) or on the shared partition (as in the diskless node case). It is recommended that they be installed on the local disk.

For definitions of the terms diskfull, diskless, and dataless nodes, see Cluster Model in the *Netra High Availability Suite 3.0 1/08 Foundation Services Overview*.

Setting Up a Makefile

To enable the compiler and linker to locate the required header files and libraries, specify the following entries in your makefile:

For the Solaris OS:

```
CFLAGS += -I/opt/SUNWcgha/include/safLDFLAGS += -  
L/opt/SUNWcgha/lib \                -R/opt/SUNWcgha/lib        -lSaC1m
```

Building a 32-bit application for use with the the Linux OS:

```
CFLAGS += -I/opt/SUNWcgha/include/safLDFLAGS += -  
L/opt/SUNWcgha/lib \                -lSaC1m
```

Building a 64-bit application for use with the the Linux OS:

```
CFLAGS += -I/opt/SUNWcgha/include/safLDFLAGS += -  
L/opt/SUNWcgha/lib \                -lSaC1m
```

Note – These entries apply only if the developer package header files and libraries are installed in their default locations. For information on installing the header files and libraries required by developers, see [“Installing Libraries and Header Files” on page 7](#).

For an example makefile that uses a specified code example, see [CODE EXAMPLE 3-4](#).

Compiling Applications

The applications you develop using the CLM API can be compiled using the Sun Studio 10 software compiler. For more information, refer to the documentation that is supplied with Sun Studio software.

Including Applications in a Startup Script

Applications that are to run on a deployed Foundation Services cluster can be started automatically when the node is booted. For this, you can supply a startup script for the application. The startup script should be located in the `/etc/init.d/` directory. Link the script to an entry in either the `/etc/rc2.d/` directory or the `/etc/rc3.d/` directory, and the script will be executed when the node boots. For more information, see the `init1M` (Solaris) or `init8` (Linux) man page.

If you require fast performance from a program, ensure that shared objects linked with the program are loaded in memory at runtime. To load shared objects in memory at runtime, set the `LD_BIND_NOW` environment variable. For more information on this variable, see the Solaris OS documentation on Runtime Linker in the *Linker and Libraries Guide*, or for the Linux OS, refer to the `ld (1)` man page.

For better performance from programs running on diskless nodes in a test cluster, you can set the `mlockall()` function within your program to lock address space. For more information, see the `mlockall3C` man page.

Running Your Applications on the Cluster

Applications that you develop on your development host can be tested on a cluster. For information about supported cluster configurations and how to connect your development host to a cluster, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Getting Started Guide*.

To transfer applications from your development host to a cluster, use one of the following commands:

<code>ftp</code>	The <code>ftp</code> command is the user interface to the Internet standard File Transfer Protocol. For more information, see the <code>ftp1</code> man page.
<code>rcp</code>	The <code>rcp</code> command is used to copy files between machines. For more information, see the <code>rcp1</code> man page.
<code>mount</code>	The <code>mount</code> command is used to mount file systems and remote resources. For more information, see the <code>mount1M</code> (Solaris) or (Linux) <code>mount8man</code> page.

Application Examples

The code fragment shown in [CODE EXAMPLE 3-1](#) enables the API to display peer node membership. This code fragment should be included when you run the CLM API examples provided in this guide. This code fragment includes `saClmClusterNodeGet()` functions, which are explained in the following chapter.

CODE EXAMPLE 3-1 Mandatory Code Fragment: `common.c`

```
void print_node(const SaClmClusterNodeT *node)
{
    printf("Node %u\n"
           "-----\n"
           "Id:      %i\n"
           "Address: IPv%c %*.s\n"
           "Name:     %*.s\n"
           "Member:    %s\n"
           "Boot TS: %lli\n\n",
           node->nodeId,
           node->nodeId,
           (node->nodeAddress.family == SA_CLM_AF_INET ? '4' : '6'),
           node->nodeAddress.value,
           node->nodeName.length, node->nodeName.length,
           node->nodeName.value,
           (node->member ? "Yes" : "No"),
           node->bootTimestamp);
}
```

The code fragment in [CODE EXAMPLE 3-1](#) is accompanied by a `common.h` header file. The code in the `common.h` header file is used in many of the code samples in this guide and is shown in [CODE EXAMPLE 3-2](#).

CODE EXAMPLE 3-2 The `common.h` Header File

```
#ifndef __CLM_COMMON__
#define __CLM_COMMON__

#include <saClm.h>

/*
 * Required version
 */
#define VERSION_RELEASE 'B'
#define VERSION_MAJOR 2
#define VERSION_MINOR 1
const char *changeString(SaClmClusterChangesT change);
```

CODE EXAMPLE 3-2 The common.h Header File (Continued)

```
const char *errString(SaAisErrorT code);

void print_node(const SaClmClusterNodeT *node);
void print_notification(const SaClmClusterNotificationT *notif);
void print_notif_buffer(const SaClmClusterNotificationBufferT
*notif);

#endif
```

For instructions on installing header files, see [“Installing Libraries and Header Files” on page 7](#).

To check that your development host and cluster are running correctly and that you are able to compile and run code using the CLM API, an example, `NodeGet.c`, is provided. This example uses:

- The `common.h` header file in [CODE EXAMPLE 3-2](#)
- The `print_node()` function from the `common.c` code fragment in [CODE EXAMPLE 3-1](#)

This worked example is shown in [CODE EXAMPLE 3-3](#).

CODE EXAMPLE 3-3 Example NodeGet.c Program

```
#include <stdio.h>
#include <stdlib.h>
#include <saClm.h>

#include "common.h"

int main(void)
{
    SaClmHandleT          clmHandle;
    SaClmClusterNodeT     clusterNode;
    SaAisErrorT           res;
    SaVersionT            version;

    version.releaseCode = VERSION_RELEASE;
    version.majorVersion = VERSION_MAJOR;
    version.minorVersion = VERSION_MINOR;

    res = saClmInitialize(&clmHandle, NULL, &version);
    if (res != SA_AIS_OK) {
        printf("Failed to initialize the library: %s\n",
            errString(res));
        exit(1);
    }
}
```

CODE EXAMPLE 3-3 Example NodeGet.c Program (Continued)

```
    }
    saClmClusterNodeGet(clmHandle, SA_CLM_LOCAL_NODE_ID,
                        SA_TIME_MAX, &clusterNode);
    if (res != SA_AIS_OK) {
        printf("Failed to get node information: %s\n",
            errString(res));
        exit(1);
    }
    print_node(&clusterNode);

    (void) saClmFinalize(clmHandle);
    return 0;
}
```

The function `errString()` does not belong to the CLM API and can be found in the `common.c` example file.

An example makefile is also provided in this section. This example makefile enables you to compile the `NodeGet.c` code in [CODE EXAMPLE 3-3](#), using:

- The `common.h` header file in [CODE EXAMPLE 3-2](#)
- The code fragment, `common.c` in [CODE EXAMPLE 3-1](#)

This example makefile is shown in [CODE EXAMPLE 3-4](#).

CODE EXAMPLE 3-4 Makefile for the NodeGet.c Program

```
CFLAGS += -I/opt/SUNWcgha/include/saf
LDLIBS += -L/opt/SUNWcgha/lib -lSaClm

PRG=NodeGet

all: $(PRG)

$(PRG): NodeGet.o common.o
    $(CC) $(LDLIBS) -o $@ NodeGet.o common.o

common.o: common.c common.h
NodeGet.o: NodeGet.c common.h

.c.o:
    $(CC) $(CFLAGS) -c $<
```

Note – The preceding example is for the Solaris OS. If your system uses the Linux OS, modify the makefile as described in [“Setting Up a Makefile” on page 10](#).

For information about setting up a makefile, see [“Setting Up a Makefile” on page 10](#).

The Foundation Services is supplied with source code examples in the `SUNWnhsafclm` developer package. These examples are installed in subdirectories of the `/opt/SUNWcgha/examples/SaClm_API` directory for Solaris-based systems, and in the `/opt/sun/nhas/examples/SaClm_API` directory for Linux-based systems.

Library Life Cycle

Before using the library, the application must initialize it by calling the `saClmInitialize()` function. This function takes two input arguments. The first argument is a pair of callback functions to be associated with the receipt of notifications and receipt of results from asynchronous API calls. The other argument is the library version requested. If the bound library is incompatible with the requested version, the call will fail.

On return, the function returns a handle to be used in any subsequent calls to the library. For more information, refer to the `saClmInitialize (3clm)` man page.

The application must obtain the selection object associated with the handle using the function `saClmSelectionObjectGet()`. Any thread tracking changes or expecting responses from asynchronous calls must wait for this selection object (a file descriptor). You can use `select(3C)` or `poll(2)` to wait for the data. When the data is available, use the `saClmDispatch()` function to process the pending callbacks.

When the library will no longer be used, call the function `saClmFinalize()` to destroy the handle and free the allocated resources.

Multiple initializations of the library are possible. Each initialization will have its own handle, and a full life cycle must be implemented for each.

Retrieving Node Information Using the CLM API

This chapter describes how to use the functions of the CLM API to retrieve information about nodes. For more information, see the following topics:

- [“Retrieving Information About a Node” on page 17](#)
 - [“Retrieving Information About All Nodes in the Cluster” on page 20](#)
 - [“Using the `saClmClusterNodeT` Structure for Information About Cluster Nodes” on page 24](#)
-

Retrieving Information About a Node

The functions `saClmClusterNodeGet()` and `saClmClusterNodeGetAsync()` retrieve information about the node specified. The node is specified by providing its node id. The special node id, `SA_CLM_LOCAL_NODE_ID`, can be used to specify the local node without knowing the local node’s node id. When run successfully, the functions return a `SaClmClusterNodeT` structure that contains information about the requested node. For information about this structure, refer to [“Using the `saClmClusterNodeT` Structure for Information About Cluster Nodes” on page 24](#).

For an example, refer to [CODE EXAMPLE 3-3](#).

The asynchronous version of the function returns immediately, but the request will be processed asynchronously by the `nhcmmmd` daemon. When the results are ready, the callback specified during the initialization of the library will be executed.

For the callback to be called, you must obtain a selection object, which you can do using the `saClmSelectionObjectGet()` function, you must use the selection object to know when a callback is pending, and you must call the `saClmDispatch()` function so callbacks are executed, as explained in [“Library Life Cycle” on page 15](#).

The following example shows the usage of the `saClmClusterNodeGetAsync()` function:

CODE EXAMPLE 4-1 Retrieving Information About the Local Node Using the `saClmClusterNodeGetAsync()` Function

```
#include <stdio.h>
#include <stdlib.h>
#include <poll.h>
#include <saClm.h>

#include "common.h"

static void node_callback(SaInvocationT invocation,
                        const SaClmClusterNodeT *clusterNode,
                        SaAisErrorT error)
{
    if (error != SA_AIS_OK) {
        printf("Callback received error %s\n",
            errString(error));
        return;
    }

    printf("Callback\n-----\nInvocation: %llu\n",
        invocation);
    print_node(clusterNode);
}

int main(void)
{
    SaClmHandleT      clmHandle;
    SaClmClusterNodeT clusterNode;
    SaAisErrorT       res;
    SaVersionT        version;
    SaClmCallbacksT   callbacks;
    struct pollfd      fds[1];
    SaSelectionObjectT selObj;

    version.releaseCode = VERSION_RELEASE;
    version.majorVersion = VERSION_MAJOR;
    version.minorVersion = VERSION_MINOR;

    callbacks.saClmClusterNodeGetCallback = node_callback;
    callbacks.saClmClusterTrackCallback = NULL;

    res = saClmInitialize(&clmHandle, &callbacks, &version);
    if (res != SA_AIS_OK) {
        printf("Failed to initialize the library: %s\n",
```


CODE EXAMPLE 4-1 Retrieving Information About the Local Node Using the
saClmClusterNodeGetAsync() Function *(Continued)*

```
        errString(res));
    exit(1);
}
res = saClmClusterNodeGetAsync(clmHandle, 1,
                               SA_CLM_LOCAL_NODE_ID);
if (res != SA_AIS_OK) {
    printf("Failed to launch NodeGetAsync: %s\n",
          errString(res));
    exit(1);
}
res = saClmSelectionObjectGet(clmHandle, &selObj);
if (res != SA_AIS_OK) {
    printf("SelectionObjectGet failed: %s\n",
          errString(res));
    exit(1);
}
fds[0].fd = (int) selObj;
fds[0].events = POLLIN;
fds[0].revents = 0;

if (poll(fds, 1, -1) != 1)
    printf("Poll failed\n");

res = saClmDispatch(clmHandle, SA_DISPATCH_ONE);
if (res != SA_AIS_OK) {
    printf("Dispatch failed: %s\n", errString(res));
    exit(1);

    (void) saClmFinalize(clmHandle);

return 0;
}
```

Note – For more information about the functions referenced in this section, refer to the corresponding man pages associated with these functions.

Retrieving Information About All Nodes in the Cluster

When called with the flag `SA_TRACK_CURRENT`, the function `saClmClusterTrack()` will retrieve information about all the cluster nodes. If the parameter `notificationBuffer` is a `NULL` pointer, the function will run asynchronously and the results will be provided to the `Track` callback function. If the pointer is not `NULL`, then the function will behave synchronously and the results will be stored in the provided buffer. In this latter case, if the `notification` field of the provided structure is a `NULL` pointer, then enough memory will be allocated to store the information. The user is responsible for freeing this memory.

In the following example, the synchronous version of the call is used and the memory is allocated by the library.

CODE EXAMPLE 4-2 Synchronous `saClmClusterTrack()` Function

```
#include <stdio.h>
#include <stdlib.h>
#include <saClm.h>

#include "common.h"

int main(void)
{
    SaClmHandleT          clmHandle;
    SaAisErrorT           res;
    SaVersionT            version;
    SaClmClusterNotificationBufferT notifBuffer;

    version.releaseCode   = VERSION_RELEASE;
    version.majorVersion  = VERSION_MAJOR;
    version.minorVersion  = VERSION_MINOR;

    res = saClmInitialize(&clmHandle, NULL, &version);
    if (res != SA_AIS_OK) {
        printf("Failed to initialize the library: %s\n",
               errString(res));
        exit 1 ;
    }

    /* Let the library allocate the memory */
    notifBuffer.numberOfItems = 0;
    notifBuffer.notification = NULL;
```

CODE EXAMPLE 4-2 Synchronous saClmClusterTrack() Function (Continued)

```

    res = saClmClusterTrack(clmHandle, SA_TRACK_CURRENT,
                           &notifBuffer);

    if (res != SA_AIS_OK) {
        printf("Track call failed: %s\n", errString(res));
        exit(1);
    }

    if (notifBuffer.notification != NULL) {
        print_notif_buffer(&notifBuffer);

        /* Free the memory allocated by the library */
        saClmClusterNotificationFree(clmHandle,
                                     notifBuffer.notification);
    }

    (void) saClmFinalize(clmHandle);

    return 0;
}

```

The function `print_notif_buffer()` is not part of the SA Forum/CLM API and can be found in the `common.c` file. The following example shows the use of the `print_notif_buffer()` function, which shows how to print the information on a `SaClmClusterNotificationBufferT` structure:

CODE EXAMPLE 4-3 The `print_notif_buffer()` function

```

void print_notification(const SaClmClusterNotificationT *notif)
{
    printf("change: %s\n", changeString(notif->clusterChange));
    print_node(&notif->clusterNode);
}

void print_notif_buffer(const SaClmClusterNotificationBufferT
*notif)
{
    int i;

    printf("Notification buffer\n"
           "-----\n");
    printf("View Number: %llu\nNumber of Items: %u\n",
           notif->viewNumber, notif->numberOfItems);
    if (notif->notification == NULL)
        printf("notification: NULL\n");
    else
        for (i = 0; i < notif->numberOfItems; i++) {
            printf("notification: %2i\n-----\n", i);
        }
}

```

CODE EXAMPLE 4-3 The `print_notif_buffer()` function (Continued)

```
        print_notification(&notif->notification[i]);  
    }  
}
```

The following example shows the asynchronous version of the call:

CODE EXAMPLE 4-4 The `saClmClusterTrackAsync()` Function

```
#include <stdio.h>  
#include <stdlib.h>  
#include <poll.h>  
#include <saClm.h>  
  
#include "common.h"  
  
static void track_callback(  
    const SaClmClusterNotificationBufferT *notificationBuffer,  
    SaUInt32T numberOfMembers,  
    SaAisErrorT error)  
{  
    if (error != SA_AIS_OK) {  
        printf("Callback received error %s\n",  
            errString(error));  
        return;  
    }  
  
    print_notif_buffer(notificationBuffer);  
}  
  
int main(void)  
{  
    SaClmHandleT      clmHandle;  
    SaAisErrorT       res;  
    SaVersionT        version;  
    SaClmCallbacksT   callbacks;  
    struct pollfd     fds[1];  
    SaSelectionObjectT selObj;  
  
    version.releaseCode = VERSION_RELEASE;  
    version.majorVersion = VERSION_MAJOR;  
    version.minorVersion = VERSION_MINOR;  
  
    callbacks.saClmClusterNodeGetCallback = NULL;  
    callbacks.saClmClusterTrackCallback = track_callback;  
  
    res = saClmInitialize(&clmHandle, &callbacks, &version);
```

CODE EXAMPLE 4-4 The saClmClusterTrackAsync() Function *(Continued)*

```
if (res != SA_AIS_OK) {
    printf("Failed to initialize the library: %s\n",
        errString(res));
    exit(1);
}

res = saClmClusterTrack(clmHandle, SA_TRACK_CURRENT, NULL);
if (res != SA_AIS_OK) {
    printf("Failed to launch async Track: %s\n",
        errString(res));
    exit(1);
}

res = saClmSelectionObjectGet(clmHandle, &selObj);
if (res != SA_AIS_OK) {
    printf("SelectionObjectGet failed: %s\n",
        errString(res));
    exit(1);
}

fds[0].fd = (int) selObj;
fds[0].events = POLLIN;
fds[0].revents = 0;

if (poll(fds, 1, -1) != 1)
    printf("Poll failed\n");

res = saClmDispatch(clmHandle, SA_DISPATCH_ONE);
if (res != SA_AIS_OK) {
    printf("Dispatch failed: %s\n", errString(res));
    exit(1);
}

        (void) saClmFinalize(clmHandle);

return 0;
}
```

Using the saClmClusterNodeT Structure for Information About Cluster Nodes

The `saClmClusterNodeT` structure, contained within the SA Forum/CLM API, is an important source of information about member nodes. This structure contains the following fields:

<i>nodeId</i>	The unique identifier of a node.
<i>nodeName</i>	The user-visible string that identifies a node and is used to format display messages.
<i>nodeAddress</i>	Stores the dotted-decimal notation of the node Carrier Grade Transport Protocol (CGTP) address in a string that can be used as a parameter on any architecture.
<i>bootTimestamp</i>	The instance of the last reboot expressed as the number of nanoseconds elapsed since 00:00:00 UTC, January 1, 1970. Nodes use this field to detect whether another node has rebooted within an interval of time.
<i>member</i>	A Boolean value indicating whether or not the node is a cluster member.
<i>initialViewNumber</i>	The view number when the node joined the cluster.

Understanding Change Notifications

This chapter describes how the CLM API indicates changes in the state of the cluster by sending notifications to system services and applications.

Introduction to Change Notifications

Notifications are information messages sent by the `nhcmmd` daemon on a node to services or applications registered to receive them. Notifications are sent when there is a change in the membership of the cluster.

Cluster notifications enable a service or application to maintain an accurate view of the state of the cluster and of the state of any peer node. An application or service can use notifications to coordinate changes in system services when a peer node joins or leaves the cluster.

To verify that the `nhcmmd` daemon is running on your peer nodes, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Cluster Administration Guide*. For information about the `nhcmmd` daemon, see the `nhcmmd1M` (Solaris) or `nhcmmd8`(Linux) man page.

Understanding the Structure of Notifications

Applications that you write can register a *callback function* to handle notification messages. The callback function `SaClmClusterTrackCallbackT` must be used and is registered during the library initialization. Notification tracking is started by calling the function `saClmClusterTrack()` with either the `SA_TRACK_CHANGES` or `SA_TRACK_CHANGES_ONLY` flag. The first flag requires notification including

information about all the cluster members, including those that left the cluster after the last notification. The second flag asks for notifications including information about only the nodes that suffered any change. The flags are mutually exclusive.

The callback function receives the following three parameters.

TABLE 5-1 Parameters of SaClmClusterTrackCallbackT Function

Parameter	Description
<i>error</i>	The error code returned by the nhcmmd daemon. This field is not used for notifications.
<i>notificationBuffer</i>	A pointer to a notification buffer structure (SaClmClusterNotificationBufferT).
<i>numberOfMembers</i>	The current number of members in the cluster.

The notification buffer has the following members.

TABLE 5-2 Members of the SaClmClusterNotificationBufferT Structure

Member	Description
<i>viewNumber</i>	The current view number of the cluster membership.
<i>notification</i>	An array of notifications.
<i>numberOfItems</i>	The number of notifications in the array.

Each element of the notification array is of the structure type SaClmClusterNotification with the following fields.

TABLE 5-3 Fields of SaClmClusterNotification Structure

Fields	Description
<i>clusterNode</i>	A structure with node information as described in “Using the saClmClusterNodeT Structure for Information About Cluster Nodes” on page 24.
<i>clusterChange</i>	The change in the cluster membership related to the node in clusterNode.

Nodes can be changed as follows:

- SA_CLM_NODE_NO_CHANGE: No change in the node.
- SA_CLM_NODE_JOINED: The node joined the cluster.
- SA_CLM_NODE_LEFT: The node left the cluster.
- SA_CLM_NODE_RECONFIGURED: Some data changed in the node.

The following example presents a simple application showing all the notifications received:

CODE EXAMPLE 5-1 Application With All Notifications Provided

```
#include <stdio.h>
#include <stdlib.h>
#include <saClm.h>

#include "common.h"

static void notif_callback(
    const SaClmClusterNotificationBufferT
    *notificationBuffer,
    SaUint32T numberOfMembers,
    SaAisErrorT error)
{
    if (error != SA_AIS_OK) {
        printf("Callback received error %s\n",
            errString(error));
        return;
    }

    print_notif_buffer(notificationBuffer);
}

int main(void)
{
    SaClmHandleT      clmHandle;
    SaAisErrorT       res;
    SaVersionT        version;
    SaClmCallbacksT   callbacks;

    version.releaseCode = VERSION_RELEASE;
    version.majorVersion = VERSION_MAJOR;
    version.minorVersion = VERSION_MINOR;

    callbacks.saClmClusterNodeGetCallback = NULL;
    callbacks.saClmClusterTrackCallback = notif_callback;

    res = saClmInitialize(&clmHandle, &callbacks, &version);
    if (res != SA_AIS_OK) {
        printf("Failed to initialize the library: %s\n",
            errString(res));
        exit(1);
    }

    res = saClmClusterTrack(clmHandle, SA_TRACK_CHANGES_ONLY,
```

CODE EXAMPLE 5-1 Application With All Notifications Provided *(Continued)*

```
        NULL);  
    if (res != SA_AIS_OK) {  
        printf("Track call failed: %s\n", errString(res));  
        exit(1);  
    }  
  
    res = saClmDispatch(clmHandle, SA_DISPATCH_BLOCKING);  
    if (res != SA_AIS_OK) {  
  
        printf("Dispatch failed: %s\n", errString(res));  
        exit(1);  
    }  
  
        (void) saClmFinalize(clmHandle);  
  
    return 0;  
}
```

In this example, the selection object is not used to poll for data. Instead, `saClmDispatch()` is called with the flag `SA_DISPATCH_BLOCKING`. Any thread calling `saClmDispatch()` with this flag remains within dispatch and executes the callbacks as they become pending. The thread does not return until the corresponding `saClmFinalize()` function is executed by another thread. Because this example has no other thread, this situation would require you to kill the process.

Applications that are configured to keep track of the cluster's state must rely on the notifications. To initially get the state of the cluster, the track function must be called with the `SA_DISPATCH_CURRENT` flag. However, between both calls, some notification could be missed. To avoid this, combine both flags in the same call to the track function.

Debugging Applications in the Foundation Services

For information about how to report and check errors caused by applications and how to debug application, see the following sections:

- [“Reporting Application Errors” on page 29](#)
- [“Reading Error Information for Debugging” on page 30](#)
- [“Stopping the Daemon Monitor for Debugging” on page 30](#)
- [“Broken Pipe Error Messages” on page 31](#)

For debugging purposes configure remote IP access to all nodes in the cluster. For more information, see “Cluster Addressing and Networking” in *Netra High Availability Suite 3.0 1/08 Foundation Services Overview*.

You can use standard Solaris Operating System commands in the Foundation Services environment. For debugging applications that interact with the Foundation Services nodes use the debugging software provided with the Sun Studio 10 software.

Reporting Application Errors

Configure applications to report errors and their causes. This information can be used during troubleshooting to reduce the risk of the re-occurrence of similar errors. To facilitate recovery from an error, you can provide the following information:

- The return value of the function call that returned the error
- The context in which the error occurred
- An indication of the severity of the error

For a list of the SA Forum/CLM API error codes, refer to the SA Forum documentation.

Reading Error Information for Debugging

In the Foundation Services, standard error and alert messages are sent to system log files. In error scenarios, you can refer to the system log files to determine the history of a process. Critical errors are written on the console in addition to being logged in the system log files.

While it is true that errors can cause notifications to be sent, notifications are events and are not errors in themselves. For information on notifications, see [“Introduction to Change Notifications” on page 25](#).

The NMA enables you to receive information on notifications. Statistics are available to diagnose the cause of errors received. See the *Netra High Availability Suite 3.0 1/08 Foundation Services NMA Programming Guide*.

Note – NMA is not available for use on the Linux platform, and is only supported for use with the Solaris OS.

For information about using and configuring system log files, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Cluster Administration Guide*.

Stopping the Daemon Monitor for Debugging

You cannot debug critical services, such as the CMM or Reliable NFS, on a running cluster. Debugging would interrupt the regular messages that these services send between nodes. Debugging tools, such as the `truss` command, cannot be used on daemons while they are being monitored by the Daemon Monitor.

Before debugging a Foundation Services daemon or a monitored Solaris daemon, stop the Daemon Monitor from monitoring the daemon that you want to debug. When you have finished debugging, restart the Daemon Monitor.

For information about how to stop and restart the Daemon Monitor, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Cluster Administration Guide*. For a list of monitored daemons, see the `nhpmd1M` (Solaris) or `nhpmd8` (Linux) man page.

Broken Pipe Error Messages

If one of the applications you are running on your cluster terminates suddenly, CMM notification pipes that this application opened are kept on the `nhcmmnd` side. You can be left with a broken pipe from the CMM to the dead application. If the CMM later sends a notification to this dead application, the CMM realizes that the application is dead and closes the broken pipe. Alternatively, the CMM frequently checks to see if a client application is dead and if necessary, closes associated pipes.

If many of your applications die suddenly, without notifying the CMM, the following can happen:

- Many pipes are broken.
- Unless the CMM has a notification to emit, neither the dead applications nor the broken pipes, are identified by the CMM.
- Each broken pipe is associated to a file descriptor. This can lead to a file descriptor shortage as the quantity of file descriptors increases, which can saturate the CMM.

If one of your applications has died suddenly, you receive a system log message such as this:

```
# Dec 23 09:56:07 machine_name CMM[839]: S-CMM
notif to /var/run/CMM_884_00000005 fails: Broken pipe
```

The CMM detects the problem and closes the notification pipe. For further information on accessing system log files, see “Accessing and Maintaining System Log Messages” in the *Netra High Availability Suite 3.0 1/08 Foundation Services Cluster Administration Guide* and the `syslog.conf4` man page.

Index

A

address space
locking, 11

APIs

- development environment, 1
- for managing peer nodes, 1
- examples of use, 12

applications

- debugging, 29
- development environment for, 5
- starting automatically, 11
- testing, 7, 11

B

binary files

- location for cluster, 9

broken pipes, 31

C

callback functions

- cluster membership change, 25

CFLAGS entry, 10

cgha_cmm library, 8

change notifications, 25

cluster, 11

- location of binary files, 9
- state, 25

Cluster Management Service

- programming interface, 1

CMM API

- examples of use, 2
- function calls, 2

header files

- accessing, 3

introduction to, 1

libraries

- accessing, 3

multithreading characteristics, 2

notifications, 25

usage, 2

cmm_cmc_filter function, 25

cmm_cmc_register function, 25

cmm_member_t structure

- fields, 24

- usage, 24

cmm_notify_t structure, 25

common.h library file

- code example, 12

compiling

- programs, 10

D

Daemon Monitor

- monitoring the nhcmmnd daemon, 8

daemons

- issues when debugging, 30

- nhcmmnd, 31

- sending notifications, 25

data transfer, 6

debugging applications, 29

development environment, 1, 5

development host

- disk space, 6

- requirements, 5

- software requirements, 6
- transferring data, 6

disks, space on development host, 6

display messages

- formatting, 24

E

environment variables

- LD_BIND_NOW, 11

errors

- checking, 29
- displaying, 12
- from function calls, 29
- log files, 30
- logging, 29
- reporting by the NMA, 30
- return values, 30

example source code, 12

F

failures

- of the active node, 1
- provision of a standby node, 1

file descriptors

- shortage of, 31

file systems

- mounting, 11

files

- copying, 11
- transferring, 11

flags

- compiler *See* Makefiles, 10

ftp command, 10

function calls

- return values, 30

H

hardware requirements, 5

- development host, 5

header files

- in SUNWnhcmdpackage, 7
- locating, 10

I

installation server

- transferring data, 6

L

LD_BIND_NOW environment variable, 11

LDFLAGS entry, 10

libcgha_cmm.so library, 8

- linking to your application, 8
- location, 8

libraries, 7

- cgha_cmm, 8
- common.h library file
- code example, 12
- libcgha_cmm.so, 8
- linking to your application, 8
- locating, 10

linking programs, 10

locking address space, 11

M

Makefiles

- CFLAGS entry, 10
- LDFLAGS entry, 10

membership roles

- change notifications, 25
- displaying, 12

memory

- locking shared objects in, 11

messages

- formatting, 24

mlockall function, 11

mount command, 10

multithreading

- characteristics of CMM API, 2

N

nhcmmnd daemon

- communicating with libcgha_cmm.so library, 8
- notification pipes, 31
- sending notifications, 25

nodes

- addr field, 24
- availability in cluster, 2
- CGTP address, 24
- diskless
- improving performance, 11
- failures, 1
- gathering information, 2
- identifying, 24

- incarnation_number field, 24
- information about, 24
- joining the cluster, 25
- last reboot, 24
- leaving the cluster, 25
- name field, 24
- nodeid field, 24
- retrieving information, 24
- standby, 1
- notifications
 - broken pipes, 31
 - callback functions, 25
 - change, 25
 - sent by nhcmmmd daemon, 25

P

- packages
 - prerequisites for CMM API, 3
 - SUNWnhcmd, 7
- performance
 - enhancing, 11
- pipes
 - broken, 31
- programs
 - compiling, 10
 - enhancing performance, 11
 - linking, 10

R

- rcp command, 10
- return values, 30

S

- saClnClusterNodeGet* functions
 - code example, 12
- shared objects
 - locking in memory, 11
- software requirements, development host, 6
- source code
 - examples, 12
- space requirements, of development host, 6
- startup scripts, 11
- Sun WorkShop TeamWare, 10
- SUNWnhcmd package, 3
 - contents, 12
- SUNWnhcmdpackage, 7

- SUNWnhhad package, 3

T

- transfer of data, 6

