



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

Sun Microsystems, Inc.
www.sun.com

Part No. 819-5236-13
March 2008, Revision A

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

Copyright 2008 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

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, Java, docs.sun.com, Solaris, Netra, and Sun Studio 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, Java, docs.sun.com, Solaris, Netra, et Sun Studio 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.



Adobe PostScript

Contents

Preface xi

- 1. Netra HA Suite Software CMM API** 1
 - Overview of CMM API 1
 - Characteristics of the CMM API 2
- 2. Introduction to the Properties of a Node** 5
 - Membership Roles 5
 - Qualification Levels 6
 - Administrative Attributes 7
- 3. Setting Up the Development Environment** 9
 - Introduction to the Development Environment 9
 - Setting Up the Development Host 10
 - Setting Up a Netra HA Suite Cluster 10
 - Installing Libraries and Header Files 11
 - ▼ To Install the Developer and Trace Packages 11
- 4. Building CMM Applications** 13
 - Installing Applications on a Cluster 13
 - Setting Up a Makefile 14

Compiling Applications	14
Including Applications in a Startup Script	15
Running Your Applications on the Cluster	15
Application Examples	16
5. Retrieving Node Information Using the CMM API	21
Identifying the Current Node	21
Retrieving Information About the Master Node or Vice-Master Node	22
Retrieving Information About Any Node	23
Retrieving Information About All Nodes in the Cluster	25
Identifying the Role of a Node	26
Identifying the Properties of a Node	27
Using the <code>cmm_member_t</code> Structure for Information About Member Nodes	28
Using the <code>sflag</code> Field of the <code>cmm_member_t</code> Structure	29
6. Understanding Change Notifications	31
Introduction to Change Notifications	31
Understanding the Structure of Notifications	32
Notification Values	33
Notifications During Changes in the Cluster State	34
Cluster Initialization Notifications	36
Vice-Master Removal Notifications	37
Vice-Master Excluded Notification	37
Peer Node Removal Notification	38
Master Node Excluded Notifications	38
Node Other Than Master Excluded Notification	38
Switchover Notifications	39
Failover Notifications	39
Failover Due to the Removal or Failure of the Master Node	39

Failover Due to Master Disqualification	40
Stale Cluster Notification	41
Amnesia	41
Split Brain	41
7. Managing Changes in the Cluster State	43
Setting a Timeout Value for Calls to the <code>nhcmmmd</code> Daemon	43
Reloading the Cluster Node Table	45
Receiving and Handling Change Notifications	45
Registering to Receive Notifications	46
Filtering Notifications	46
Receiving and Dispatching Notifications	47
Retrieving Change Notifications	47
Responding to Cluster Notifications by Modifying the Cluster	49
Removing or Excluding a Node	49
Removing the Master Node	50
Removing the Vice-Master Node	50
Removing Diskless and Dataless Nodes	51
Setting the Qualification of a Node	51
Triggering a Switchover	51
Triggering a Switchover Using <code>cmm_mastership_release()</code>	52
Triggering a Failover	53
Triggering a Failover by Using the <code>cmm_membership_remove()</code> Function	53
Triggering a Failover by Using the <code>cmm_member_setqualif()</code> Function	55
Rejoining the Cluster	55
8. Debugging Applications in the Foundation Services	59
Reporting Application Errors	59

Reading Error Information for Debugging	60
Stopping the Daemon Monitor for Debugging	60
Broken Pipe Error Messages	61
Return Values of the CMM API	62
Index	65

Figures

FIGURE 1-1 Interaction of the CMM With Your Applications Using the CMM API 2

FIGURE 3-1 Setting Up the Development Environment 10

Code Examples

- CODE EXAMPLE 4-1 `common.c` 16
- CODE EXAMPLE 4-2 The `common.h` Header File 17
- CODE EXAMPLE 4-3 Example `master_node.c` Program 17
- CODE EXAMPLE 4-4 Makefile for the `master_node.c` Program 18
- CODE EXAMPLE 5-1 `current_node.c` 22
- CODE EXAMPLE 5-2 `vicemaster_node.c` 23
- CODE EXAMPLE 5-3 Retrieving Information About the Current Node Using the `cmm_member_getinfo()` Function 24
- CODE EXAMPLE 5-4 `all_nodes.c` 25
- CODE EXAMPLE 6-1 The `cmm_cmc_notification_t` Structure 32
- CODE EXAMPLE 7-1 `connect.c` 44
- CODE EXAMPLE 7-2 `notif.c` 47
- CODE EXAMPLE 7-3 Disqualifying a Node 51
- CODE EXAMPLE 7-4 `switchover.c` 52
- CODE EXAMPLE 7-5 `failover.c` 54
- CODE EXAMPLE 7-6 `join.c` 56

Preface

This book describes how to write applications that use the Cluster Membership Manager API for the Netra™ High Availability (HA) Suite 3.0 1/08 Foundation Services.

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 by using the CMM API
-

Before You Read This Book

To write applications for Netra HA Suite, 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 CMM API.
- [Chapter 2](#) describes the roles, qualification levels, and attributes that a node can have.
- [Chapter 3](#) describes the requirements of a development host on which to write applications that use the CMM API.
- [Chapter 4](#) describes how to install, compile, and run your applications.
- [Chapter 5](#) describes how to identify and retrieve data about nodes in the cluster.
- [Chapter 6](#) describes the `nhcmmnd` daemon, the notifications this daemon sends, and how to interpret these notifications.
- [Chapter 7](#) describes how to retrieve and react to cluster notifications, and how to modify the cluster if necessary.
- [Chapter 8](#) describes how to debug your applications. This chapter also describes the return values provided by the CMM 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
<i>AaBbCc123</i>	The names of commands, files, and directories; on-screen computer output	Edit your <i>.login</i> file. Use <i>ls -a</i> 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 rm <i>filename</i> .

Note – Characters display differently depending on browser settings. If characters do not display correctly, change the character encoding in your browser to Unicode UTF-8.

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
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 CMM Programming Guide,
part number 819-5236-13

Netra HA Suite Software CMM API

You can use the CMM API to develop programs for highly available clusters running the Foundation Services. For more information, see the following topics:

- [“Overview of CMM API” on page 1](#)
- [“Characteristics of the CMM API” on page 2](#)

Overview of CMM API

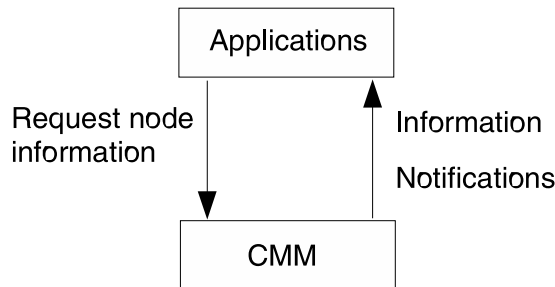
The CMM API provides a programming interface to the Cluster Membership Manager (CMM). For information about the CMM, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Overview*.

The CMM API enables you to write applications that manage peer nodes in the cluster and that monitor the role, qualification level, and state of these peer nodes. You can use the CMM API to obtain information about which node is the master node, the vice-master node, and a member of the cluster.

The CMM API also provides functions that modify the state of a cluster and enable a node to receive information about changes in the state of a cluster.

This interaction of the CMM API with the Netra HA Suite software is illustrated in [FIGURE 1-1](#).

FIGURE 1-1 Interaction of the CMM With Your Applications Using the CMM API



Characteristics of the CMM API

The CMM API enables you to create highly available applications to determine which nodes are in the cluster, and which of these nodes is the master node and the vice-master node. The main functions of the CMM API are to:

- Manage the membership of peer nodes
- Determine the availability of peer nodes
- Provide a failover framework for critical applications
- Gather information about some or all peer nodes

The CMM 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.
- The CMM 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.

Examples of the CMM API are provided in the `SUNWnhas-cmm-headers` for the Solaris OS package and `sun-nhas-cmm-headers-3.0-*.x86_amd64.rpm` for the Linux OS. These examples are available after installation on a peer node, in the `/opt/SUNWcgha/examples/cmm_API` directory for the Solaris OS or the `/opt/sun/nhas/examples/cmm_API` directory for the Linux OS.

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

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

To access the CMM API header file and libraries on the Solaris OS, install the `SUNWnhas-cmm-headers` and `SUNWnhas-cmm-libs` packages in your development environment.

- 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 CMM API header file and libraries on the Linux OS, install the `sun-nhas-cmm-headers-3.0-*.x86_amd64.rpm` and `sun-nhas-cmm-libs-3.0-*.x86_amd64.rpm` packages in your development environment.

See [Chapter 3](#) and [Chapter 4](#) for information about creating a development environment for the Foundation Services product.

Introduction to the Properties of a Node

This chapter introduces the roles, qualification levels and attributes of nodes. For information about the types of errors that occur on certain nodes, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Troubleshooting Guide*.

- [“Membership Roles” on page 5](#)
- [“Qualification Levels” on page 6](#)
- [“Administrative Attributes” on page 7](#)

Membership Roles

Peer nodes can be recognized by the CMM API as having the following membership roles:

Master node	A node with membership role <code>CMM_MASTER</code> . This node coordinates all of the cluster membership information. The master node has the current view of the cluster configuration and all the nodes receive the cluster view from the master node.
Vice-master node	A node with membership role <code>CMM_VICEMASTER</code> . The vice-master node can take over the master role of the cluster. The vice-master node has its own copy of the <code>cluster_nodes_table</code> file. For more information, see the <code>cluster_nodes_table(4)</code> man page.
Out node	Any node configured to be in the cluster that has the membership role <code>CMM_OUT_OF_CLUSTER</code> . This role means that the node is not available for use, for either physical or administrative reasons. Do not distribute tasks on an out node, because it might be undergoing maintenance.

No membership role is assigned to a node running the Foundation Services services and fully participating in cluster communication that is neither a master node nor a vice-master node. Further information about nodes in the cluster is provided by the `cluster_nodes_table(4)` man page.

For information about the definitions of the roles of nodes in the cluster, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Overview*.

Membership roles are dynamic and are defined by the master node. Unless the membership role of a node is `CMM_OUT_OF_CLUSTER`, the node is by default viewed as being in the cluster. While the node has the `CMM_OUT_OF_CLUSTER` role, the qualification level and `CMM_FLAG_SYNCHRO_NEEDED` flag are meaningless. Information about the membership role of a node can be found in the `sflag` field of the `cmm_member_t` structure. See [“Using the `sflag` Field of the `cmm_member_t` Structure” on page 29](#).

Qualification Levels

The qualification level of a node is applicable only to master-eligible nodes. On these nodes, the qualification level determines whether the node can participate in an election for the master role or vice-master role. A master-eligible node can be qualified or disqualified. The qualification levels of a master-eligible node are:

<code>CMM_QUALIFIED_MEMBER</code>	The node is qualified to be master. Only meaningful for eligible nodes.
<code>CMM_DISQUALIFIED_MEMBER</code>	The node is disqualified from being master. Only meaningful for eligible nodes.

For master-eligible nodes, the qualification level is stored in the minimum configuration file, `target.conf`, on the node and in the cluster node table. For more information, see the `target.conf(4)` man page. The qualification level is persistent, that is, if the node is rebooted, the node starts with the same qualification level it had before the reboot. The qualification level of a node can be changed during runtime. Diskless and dataless nodes are never assigned qualification levels.

To assign a new qualification level to a node, use the `cmm_member_setqualif` function. For more information, see the `cmm_member_setqualif(3CMM)` man page.

An example in which this function is used to trigger a failover in the cluster is provided in this book. See [“Triggering a Failover by Using the `cmm_member_setqualif\(\)` Function” on page 55](#).

Administrative Attributes

The CMM API recognizes each node in the cluster as having an administrative attribute. An attribute can be any of the following:

<code>CMM_ELIGIBLE_MEMBER</code>	The node is a member of the cluster. The node is a master-eligible node, therefore it is diskfull and can participate in a master or vice-master election.
<code>CMM_FLAG_DISQUALIFIED</code>	The node is part of the cluster and is master-eligible, but this node cannot participate currently in master elections. This flag applies only to master-eligible nodes.
<code>CMM_FLAG_SYNCHRO_NEEDED</code>	<p>The master node disk and vice-master node disk must be synchronized so that the vice-master node can take over the role of master node if necessary. If the disks are not synchronized, the vice-master node cannot become the master node. If the <code>CMM_FLAG_SYNCHRO_NEEDED</code> flag is set, the vice-master node disk is not up-to-date. When the master node disk and vice-master node disks are synchronized, the flag is cleared. The flag applies only to master-eligible nodes, and must not be set by applications.</p> <p>For more information about Reliable NFS, see the <i>Netra High Availability Suite 3.0 1/08 Foundation Services Overview</i>.</p>

A master node can be demoted because of a change in its administrative attributes, when the vice-master detects a problem on the master, or after a call to the `cmm_mastership_release()` function. The master node can also be demoted in the case of a failover. See [“Failover Notifications” on page 39](#). A node assumes the master role if it is sufficiently qualified.

Information about the administrative attributes of a node can be found in the `sflag` field of the `cmm_member_t` structure. For more information, see [“Using the `sflag` Field of the `cmm_member_t` Structure” on page 29](#).

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:

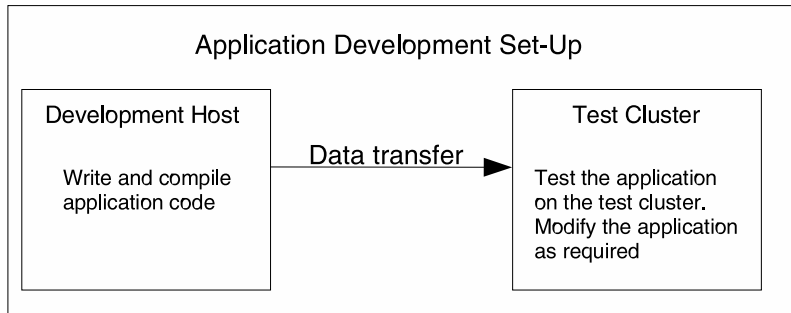
- [“Introduction to the Development Environment” on page 9](#)
- [“Setting Up the Development Host” on page 10](#)
- [“Setting Up a Netra HA Suite Cluster” on page 10](#)
- [“Installing Libraries and Header Files” on page 11](#)

For general information about developing applications on the Solaris Operating System, see the *Solaris Software Developer Collection* for the version of the Solaris OS in use at your site.

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 Netra HA Suite software 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 3-1 Setting Up the Development Environment



The development environment refers to the setup 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 Netra HA Suite 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 how to set up a cluster that runs the Netra HA Suite software, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Getting Started Guide*.

To install, compile, and run your applications, see [Chapter 4](#).

Installing Libraries and Header Files

Ensure that the library contents and header files of the Netra HA Suite software are available in the runtime environment of your development host. This includes the CMM header (.h) files and library (.so) files. These files are delivered as follows:

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

You can write your applications on your development host using the CMM 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 and Trace Packages

1. Log into the development host as superuser.
2. Install the packages as follows:

On 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 OS:

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

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

3. **Verify that the `libcgha_cmm.so` library is available in the `/opt/SUNWcgha/lib/` directory (Solaris OS), the `/opt/sun/lib` directory (32-bit applications on Linux OS), or the `/opt/sun/lib64` directory (64-bit applications on Linux OS) and that the header files are available in the `/opt/SUNWcgha/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 CMM API is provided by the `libcgha_cmm.so` library. The `libcgha_cmm` library communicates with the `nhcmmmd` daemon, which is monitored by the Daemon Monitor, `nhpmd`. For further information on the `nhcmmmd` daemon, see the `nhcmmmd(1M)` man page. For more information on the Daemon Monitor, see the `nhpmd(1M)` man page.

4. **To link the `libcgha_cmm` library to your application, add the `/opt/SUNWcgha/lib/` directory (Solaris OS), the `/opt/sun/lib` directory (32-bit applications on Linux OS), 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 Netra HA Suite Cluster”](#) on page 10.

Building CMM Applications

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

- “Installing Applications on a Cluster” on page 13
- “Setting Up a Makefile” on page 14
- “Compiling Applications” on page 14
- “Including Applications in a Startup Script” on page 15
- “Running Your Applications on the Cluster” on page 15
- “Application Examples” on page 16

Installing Applications on a Cluster

If your application is to be run on a master-eligible 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 either on the local disk (as in the master-eligible node case) or on the shared partition (as in the diskless node case), but they should 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/includeLDFLAGS += -L/opt/SUNWcgha/lib \
-R/opt/SUNWcgha/lib
```

For the Linux OS:

- Building a 32-bit application:

```
CFLAGS += -I/opt/sun/includeLDFLAGS += -L/opt/sun/lib
```

- Building a 64-bit application:

```
CFLAGS += -I/opt/sun/includeLDFLAGS += -L/opt/sun/lib64
```

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 11](#).

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

Compiling Applications

The applications you develop using the CMM API can be compiled using the Sun™ Studio software compiler. For more information, refer to the documentation 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 `init(1M)` man page.

If you require fast performance from a program, ensure that shared objects linked with the program are bound at startup. To prebind the shared objects, set the `LD_BIND_NOW` environment variable. For more information on this variable, see the Solaris documentation about Runtime Linker *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, you can set the `mlockall()` function within your program to lock address space. For more information, see the `mlockall(3C)` 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>ftp(1)</code> man page.
<code>rcp</code>	The <code>rcp</code> command is used to copy files between machines. For more information, see the <code>rcp(1)</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>mount(1M)</code> man page, or for the Linux OS, refer to the <code>mount(8)</code> man page.

Application Examples

The code fragment shown in [CODE EXAMPLE 4-1](#) provides functions to display peer node membership information. This code is provided in the module `common.c` of the examples `c` and should be included when you run the CMM API examples provided in this guide. These functions use the set of functions `cmm_member_isXXXX()`, which are explained in [Chapter 5](#).

CODE EXAMPLE 4-1 `common.c`

```
#include <stdio.h>
#include <cmm/cmm.h>

#include "common.h"

static const char *get_role_str(cmm_member_t const *P_Member)
{
    const char *L_Result;

    if (cmm_member_ismaster(P_Member))
        L_Result = "MASTER";
    else if (cmm_member_isvicemaster(P_Member))
        L_Result = "VICE-MASTER";
    else if (cmm_member_isoutofcluster(P_Member))
        L_Result = "OUT";
    else
        L_Result = "IN";

    return L_Result;
}

void print_member (cmm_member_t const *P_Member)
{
    printf("-----\n");
    printf("node_id      = %d\n", P_Member->nodeid);
    printf("domain_id    = %d\n", P_Member->domainid);
    printf("name         = %s\n", P_Member->name);
    printf("role         = %s\n", get_role_str(P_Member));
    printf("qualified    = %s\n",
        (cmm_member_isdisqualified(P_Member)) ? "NO" : "YES");
    printf("synchro.     = %s\n",
        (cmm_member_isdesynchronized(P_Member)) ? "NEEDED" :
        "READY");
    printf("frozen       = %s\n",
        (cmm_member_isfrozen(P_Member)) ? "NO" : "YES");
    printf("excluded     = %s\n",
```


CODE EXAMPLE 4-1 common.c (Continued)

```

        (cmm_member_isexcluded(P_Member)) ? "NO" : "YES");
    printf("eligible      = %s\n",
        (cmm_member_iseligible(P_Member)) ? "NO" : "YES");
    printf("incarn.       = %d\n", P_Member->incarnation_number);
    printf("swload_id    = %s\n", P_Member->software_load_id);
    printf("CGTP @       = %s\n", P_Member->addr);
    printf("init.election = %d\n", P_Member->init.election);
    printf("-----\n");
}

```

CODE EXAMPLE 4-1 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 4-2**.

CODE EXAMPLE 4-2 The common.h Header File

```

#ifndef __CMM_COMMON__
#define __CMM_COMMON__

#include <cmm/cmm.h>

extern void print_member(cmm_member_t const *P_Member);

#endif

```

For instructions on installing header files, see [“Installing Libraries and Header Files” on page 11](#).

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

- The common.h header file in **CODE EXAMPLE 4-2**
- The print_member() function from the common.c code fragment in **CODE EXAMPLE 4-1**

CODE EXAMPLE 4-3 shows a sampling of this program.

CODE EXAMPLE 4-3 Example master_node.c Program

```

#include <stdio.h>
#include <cmm/cmm.h>

#include "common.h"

int main(void)

```

CODE EXAMPLE 4-3 Example master_node.c Program (Continued)

```
{
    cmm_member_t member_info;
    cmm_error_t result;

    result = cmm_master_getinfo(&member_info);
    if (result != CMM_OK) {
        printf("Couldn't get master node information: %s\n",
               cmm_strerror(result));
    }
    else
        print_member(&member_info);
    return (result == CMM_OK ? 0 : 1);
}
```

An example makefile is also provided in this section. This example makefile enables you to compile the master_node.c code in [CODE EXAMPLE 4-3](#), using:

- The common.h header file in [CODE EXAMPLE 4-2](#)
- The common.c in [CODE EXAMPLE 4-1](#)

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

CODE EXAMPLE 4-4 Makefile for the master_node.c Program

```
NHAS_DIR = /opt/SUNWcgha

CFLAGS = -I$(NHAS_DIR)/include
LDFLAGS = -L$(NHAS_DIR)/lib \
          -R$(NHAS_DIR)/lib \
          -lcgha_cmm

all: master_node

master_node: master_node.o common.o
    $(CC) $(LDFLAGS) -o $@ master_node.o common.o

master_node.o: master_node.c common.h

common.o: common.c common.h

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

clean:
    rm -rf *.o master_node
```

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

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 14](#).

The Netra HA Suite software is supplied with source code examples in the SUNWnhcmd developer package. These examples are installed in subdirectories of the /opt/SUNWcgha/examples/ directory for Solaris-based systems and in the /opt/sun/nhas/examples/ directory for Linux-based systems.

Retrieving Node Information Using the CMM API

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

- “Identifying the Current Node” on page 21
- “Retrieving Information About the Master Node or Vice-Master Node” on page 22
- “Retrieving Information About Any Node” on page 23
- “Retrieving Information About All Nodes in the Cluster” on page 25
- “Identifying the Role of a Node” on page 26
- “Identifying the Properties of a Node” on page 27
- “Using the `cmm_member_t` Structure for Information About Member Nodes” on page 28

Identifying the Current Node

The `cmm_node_getid()` function, described in the `cmm_node_getid(3CMM)` man page, retrieves the *nodeid* of the current node, that is, the node on which your application is currently running. The *nodeid* is used in other CMM calls as a parameter.

If the current node is a nonpeer node, the `cmm_node_getid()` function returns the `CMM_ECONN` message. For further details on return values, see [TABLE 8-1](#).

The following example demonstrates how to use the `cmm_node_getid()` function:

CODE EXAMPLE 5-1 `current_node.c`

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

int main(void)
{
    cmm_nodeid_t curr_node;
    cmm_error_t result;

    result = cmm_node_getid(&currnode)
    if (result != CMM_OK
        printf("Couldn't get node id: %s\n",
            cmm_strerror(result));
    else
        printf("Current node id: %u\n", curr_node);
    return (result == CMM_OK ? 0 : 1);
}
```

Retrieving Information About the Master Node or Vice-Master Node

The `cmm_master_getinfo()` function retrieves all of the available information about the master node in the cluster. This is similar to the `cmm_member_getinfo()` function, but you do not need to provide the *nodeid*.

If there is no master node, the cluster is not in a valid state and the call returns the `CMM_ENOCLUSTER` error. During a failover triggered by the failure or disqualification of the master node, or during the cluster startup, there is a time during which no master exists. During this period, the `CMM_ENOCLUSTER` error is returned.

For more information about return values, see [“Return Values of the CMM API” on page 62](#).

The `cmm_vicemaster_getinfo()` function retrieves all of the available information about the vice-master node in the cluster. If there is no vice-master node, the function returns the `CMM_ESRCH` error. For information, see the `cmm_vicemaster_getinfo(3CMM)` man page.

These functions return the `cmm_member_t` structure. For information about the `cmm_member_t` structure, see [“Using the `cmm_member_t` Structure for Information About Member Nodes” on page 28](#).

The following example demonstrates how to test for the presence of a vice-master node by using the `cmm_vicemaster_getinfo()` function.

CODE EXAMPLE 5-2 `vicemaster_node.c`

```
#include <stdio.h>
#include <cmm/cmm.h>

#include "common.h"

int main(void)
{
    cmm_member_t  member_info;
    cmm_error_t   result;

    result = cmm_vicemaster_getinfo(&member_info);
    switch (result) {
    case CMM_OK:
        printf("Current vicemaster node is: \n");
        print_member(&member_info);
        break ;
    case CMM_ESRCH:
        printf("No vicemaster node currently in the cluster\n");
        break ;
    default:
        printf("Couldn't get vicemaster node information: %s\n",
            cmm_strerror(result));
        break;
    }
    return (result == CMM_OK ? 0 : 1);
}
```

Retrieving Information About Any Node

The `cmm_member_getinfo()` and `cmm_potential_getinfo()` functions retrieve information about a specified node as explained in [“Identifying the Properties of a Node” on page 27](#). The node must be identified by the `nodeid` argument. These functions return the `cmm_member_t` structure. For more information, see [“Using the `cmm_member_t` Structure for Information About Member Nodes” on page 28](#).

If the `cmm_member_getinfo()` function returns the `CMM_ESRCH` error, the node is out of the cluster. The `cmm_potential_getinfo()` function will not fail in this situation and the member's membership role will be `CMM_OUT_OF_CLUSTER`.

For more information about these functions, see the `cmm_member_getinfo(3CMM)` and `cmm_potential_getinfo(3CMM)` man pages.

The following code example demonstrates how to retrieve information about the current node by using the `cmm_member_getinfo()` function:

CODE EXAMPLE 5-3 Retrieving Information About the Current Node Using the `cmm_member_getinfo()` Function

```
#include <stdio.h>
#include <cmm/cmm.h>

#include "common.h"

int main(void)
{
    cmm_member_t  member_info;
    cmm_error_t   result;
    cmm_nodeid_t  current;

    result = cmm_member_getinfo(current, &member_info);
    if (result != CMM_OK) {
        printf("Failed to retrieve the current node's id:\n %s",
            cmm_strerror(result));
        return 1;
    }

    result = cmm_member_getinfo(current, &member_info);
    switch (result) {
    case CMM_OK:
        printf("Current member node is:\n");
        print_member(&member_info);
        break;
    case CMM_ESRCH:
        printf("Current node is NOT in the cluster\n");
        break;
    default:
        printf("Couldn't get member node information: %s\n",
            cmm_strerror(result));
        break;
    }
    return (result == CMM_OK ? 0 : 1);
}
```

Retrieving Information About All Nodes in the Cluster

The `cmm_member_getcount()` and `cmm_member_getall()` functions retrieve information for all peer nodes in the cluster.

- The `cmm_member_getcount()` function counts the number of peer nodes in the cluster.
- The `cmm_member_getall()` function fills an array of `cmm_member_t` structures.

Using the value returned by the `cmm_member_getcount()` function, you can dynamically allocate the table. For more information, see the `cmm_member_getcount(3CMM)` man page.

The following example demonstrates how to retrieve information about all peer nodes in the cluster.

CODE EXAMPLE 5-4 all_nodes.c

```
#include <stdio.h>
#include <stdlib.h>
#include <cmm/cmm.h>

#include "common.h"

int main(void)
{
    cmm_member_t *member_table;
    cmm_error_t result;
    uint32_t member_count;
    uint32_t members_in_table;
    uint32_t i;

    result = cmm_member_getcount(&member_count);
    if (result != CMM_OK) {
        printf("Failed to get the node count: %s\n",
              cmm_strerror(result));
        return 1;
    }

    member_table = (cmm_member_t *) calloc(member_count,
                                           sizeof(cmm_member_t));

    if (member_table == NULL) {
```

CODE EXAMPLE 5-4 all_nodes.c (Continued)

```
        printf("Failed to allocate memory\n");
        return 1;
    }

    result = cmm_member_getall(member_count, member_table,
                               &members_in_table);
    if (result != CMM_OK) {
        printf("Failed to get the node's information: %s\n",
              cmm_strerror(result));
        return 1 ;
    }

    for (i = 0; i < members_in_table; i++)
        print_member(&member_table[i]);

    return 0;
}
```

Identifying the Role of a Node

The `cmm_potential_getinfo()` and `cmm_member_getinfo()` functions retrieve information about a node identified by its *nodeid*. For details on the roles that a node can have, see [“Membership Roles” on page 5](#).

To find the *nodeid* of a node, see [CODE EXAMPLE 5-1](#).

The following functions retrieve specific information about the role of a node:

<code>cmm_member_ismaster()</code>	This function tests whether the node identified by the <code>cmm_member_t</code> structure is master. See the <code>cmm_member_ismaster(3CMM)</code> man page.
<code>cmm_member_isvicemaster()</code>	This function tests whether the node identified by the <code>cmm_member_t</code> structure is vice-master. See the <code>cmm_member_isvicemaster(3CMM)</code> man page.
<code>cmm_member_isoutofcluster()</code>	This function tests whether a node is out of the cluster. See the <code>cmm_member_isoutofcluster(3CMM)</code> man page.

If the tested condition is false, the functions in the preceding list return 0. Otherwise, these functions return a value other than 0. These functions are used in [CODE EXAMPLE 4-1](#).

You can also find the role of a node from the command line by using the `nhcmmrole` command on the node. For details about this command, see the `nhcmmrole(1M)` man page, or for the Linux OS, refer to the `nhcmmrole(8)` man page.

Note – The role of a node is also specified in the `sflag` field of the `cmm_member_t` structure returned by these functions. See [“Using the `sflag` Field of the `cmm_member_t` Structure” on page 29](#). To get node information, it is better to use the CMM API. Do not attempt direct extraction of node information from the `sflag` field of the `cmm_member_t` structure.

Identifying the Properties of a Node

The eligibility of a node to become master is determined by properties such as its master-eligibility, qualification level, and synchronization state. The qualification level of a node, relevant only for master-eligible nodes, determines if the node can participate in a master or vice-master election. For information about qualification levels that a node can have, see [“Qualification Levels” on page 6](#). The following functions retrieve information about the eligibility and qualification level of a node:

- `cmm_member_getinfo()`
- `cmm_potential_getinfo()`
- `cmm_member_iseligible()`
- `cmm_member_isdesynchronized()`
- `cmm_member_isqualified()`
- `cmm_member_isdisqualified()`

The `cmm_member_getinfo()` function returns information for any peer node that is master, vice-master, or in the cluster.

The `cmm_potential_getinfo()` function returns information for all peer nodes, even if the node has the `CMM_OUT_OF_CLUSTER` role and has not yet entered the cluster.

If a node is disqualified, it cannot become master or vice-master until it is requalified. Further information can be found in the relevant man pages, such as `cmm_member_iseligible(3CMM)`. For an example that demonstrates how to requalify a node, see [“Setting the Qualification of a Node” on page 51](#).

The function `cmm_member_isdesynchronized()` tests if the master and vice-master nodes are synchronized. If the master and vice-master nodes are not synchronized, the vice-master node cannot become the master if the master fails. Similarly, at cluster startup, a desynchronized node cannot be elected master even if

it is master-eligible and qualified. For more information about synchronization, see the definition of `CMM_FLAG_SYNCHRO_NEEDED` in [“Administrative Attributes” on page 7](#). For more information about the `cmm_member_isdesynchronized()` function, see the `cmm_member_isdesynchronized(3CMM)` man page.

The `cmm_member_getinfo()` and `cmm_potential_getinfo()` functions retrieve information about a node identified by *nodeid*. To find the *nodeid*, see [CODE EXAMPLE 5-1](#). Both functions retrieve this information from the `cmm_member_t` structure. For information about these functions, see the `cmm_member_getinfo(3CMM)` and `cmm_potential_getinfo(3CMM)` man pages.

Using the `cmm_member_t` Structure for Information About Member Nodes

The `cmm_member_t` structure, contained within the CMM 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>name</i>	The user-visible string that identifies a node and is used to format display messages.
<i>addr</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. The size of this string is sufficient for IPv4 and IPv6 addresses.
<i>incarnation_number</i>	The instant of the last reboot expressed as the number of seconds elapsed since 00:00:00 UTC, January 1, 1970. The <i>incarnation_number</i> is computed locally by every node and sent to the master node, which dispatches it. Nodes use this field to detect whether another node has rebooted within an interval of time.
<i>sflag</i>	State information about the node. This is a concatenation of the administrative attributes, membership role, and qualification levels. For more detailed information about the <i>sflag</i> field, see “Using the sflag Field of the <code>cmm_member_t</code> Structure” on page 29 .

<i>domainid</i>	The unique ID of the cluster that the node can join or has joined. All peer nodes in a cluster have the same <i>domainid</i> . A node can belong to only one cluster and the ID of this cluster is the cluster <i>domainid</i> of the current node.
<i>software_load_id</i>	This field is set at 1.
<i>initial_election</i>	The election number when the node joined the cluster. The election number is increased with each change in the cluster state. This means that two nodes that joined the cluster at the same time can have the same initial election number. However, when two nodes join the cluster at different times, the second one will have an initial election number higher than that of the first node.

Using the `sflag` Field of the `cmm_member_t` Structure

As explained in [“Using the `cmm_member_t` Structure for Information About Member Nodes” on page 28](#), the `sflag` part of the `cmm_member_t` structure stores information about a node's administrative attributes, membership role, and qualification levels. This information is stored in a bit mask in the `sflag`. The following functions extract information from the `sflag` field:

<code>cmm_member_isdesynchronized()</code>	Determines whether a master-eligible node is desynchronized. While the node has the <code>CMM_OUT_OF_CLUSTER</code> role, the qualification level and <code>CMM_FLAG_SYNCHRO_NEEDED</code> flag are meaningless.
<code>cmm_member_isdisqualified()</code>	Determines whether a master-eligible node has the <code>CMM_DISQUALIFIED_MEMBER</code> qualification level. While the node has the <code>CMM_OUT_OF_CLUSTER</code> role, the qualification level and <code>CMM_FLAG_SYNCHRO_NEEDED</code> flag are meaningless.
<code>cmm_member_isqualified()</code>	Determines whether a master-eligible node has the <code>CMM_QUALIFIED_MEMBER</code> qualification level. While the node has the <code>CMM_OUT_OF_CLUSTER</code> role, the qualification level and <code>CMM_FLAG_SYNCHRO_NEEDED</code> flag are meaningless.
<code>cmm_member_iseligible()</code>	Determines whether a node has the <code>CMM_ELIGIBLE_MEMBER</code> attribute.
<code>cmm_member_isexcluded()</code>	Determines whether a node has the <code>CMM_EXCLUDED_MEMBER</code> attribute.
<code>cmm_member_isfrozen()</code>	Determines whether a node has a <code>CMM_FROZEN_MEMBER</code> attribute.

<code>cmm_member_isoutofcluster()</code>	Determines whether a node has the <code>CMM_OUT_OF_CLUSTER</code> role. While a node has the <code>CMM_OUT_OF_CLUSTER</code> role, the qualification level and synchronization flag of the node are meaningless.
<code>cmm_member_isvicemaster()</code>	Determines whether a node has the <code>CMM_VICEMASTER</code> role.
<code>cmm_member_ismaster()</code>	Determines whether a node has the <code>CMM_MASTER</code> role.

Note – It is safer to use the `cmm_member_is...()` functions than to rely on direct extraction of node information from these extract flags.

For more information about the `cmm_member_is...()` functions, see [“Identifying the Current Node” on page 21](#), and [“Identifying the Role of a Node” on page 26](#). See also the `cmm_member_iseligible(3CMM)` and `cmm_member_ismaster(3CMM)` man pages.

Understanding Change Notifications

This chapter describes how the CMM API indicates changes in the state of the cluster by sending notifications to system services and applications. For more information, see the following topics:

- [“Introduction to Change Notifications” on page 31](#)
- [“Notifications During Changes in the Cluster State” on page 34](#)

Introduction to Change Notifications

Notifications are information messages sent by the `nhcmmmd` 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.

In a cluster, the master node is aware of all changes in the state of peer nodes. The cluster state information held by the `nhcmmmd` daemon on the master node is propagated to all peer nodes.

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.

A single change in the cluster state can cause an application or service to receive several associated cluster change notifications. This can be due to the fact that a change in the membership of one node can effect changes in the membership of several other nodes.

A cluster change notification does not contain any information about the previous role of a node. Therefore, for example, when a callback is invoked with the `CMM_MEMBER_LEFT` notification, the indicated node could have been in the cluster with no role, or could have had the `CMM_MASTER` or `CMM_VICE_MASTER` role.

Several scenarios in which there are changes in the state of the cluster, and the associated notifications sent during these changes, are described in [“Notifications During Changes in the Cluster State” on page 34](#).

For an example of how to retrieve notifications about changes in the cluster state, see [CODE EXAMPLE 7-2, “notif.c” on page 47](#).

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 `nhcmmd(1M)` man page, or for the Linux OS, refer to the `nhcmmd(8)` man page.

Understanding the Structure of Notifications

Applications that you write can register a *callback function* to handle notification messages. The `cmm_notify_t` callback receives the cluster membership change (`cmc`) callback function. The `cmm_cmc_register()` function takes the service or application data and this callback function. You must provide relevant data when registering. The code in [CODE EXAMPLE 6-1](#) details the related structure, called the `cmm_cmc_notification_t` structure.

CODE EXAMPLE 6-1 The `cmm_cmc_notification_t` Structure

```
typedef struct {
    cmm_cmchanges_t  cmchange;
    cmm_nodeid_t     nodeid;
} cmm_cmc_notification_t;
```

The fields in this structure detail the cluster change and specify the node concerned. These fields are described in [TABLE 6-1](#).

TABLE 6-1 Description of Fields of the `cmm_cmc_notification_t` Structure

Field	Description
<i>nodeid</i>	This field represents the node on which the change occurs if the notification is made for each node.
<i>cmchange</i>	This field indicates the type of change that occurs.

This structure is used by the `cmm_notify_t()` callback function. The `cmm_notify_t()` callback function contains the parameters described in [TABLE 6-2](#).

TABLE 6-2 Description of the Parameters of the `cmm_notify_t()` Callback Function

Parameter	Description
<i>change_notification</i>	This parameter is a pointer to a structure describing the specific membership change and the affected cluster member's identity. This is either the <i>nodeid</i> of a specific node, or 0 when the change affects all peer nodes.
<i>client_data</i>	This parameter is a service or application-defined value given to the <code>cmm_cmc_register()</code> function. The CMM API does not use this parameter internally.

If change notification data is required for longer than the duration of the callback, it must be handled by the client application or service.

Notification Values

Change notification messages contain the *nodeid* of the affected node and a `cmm_cmchanges_t` data type. The `cmm_cmchanges_t` data type describes the change notification. [TABLE 6-3](#) lists the notifications of the `cmm_cmchanges_t` structure:

TABLE 6-3 Change Notifications

Value	Description
<code>CMM_INVALID_CLUSTER</code>	A critical problem occurred. For example, there are two master nodes. One node must be rebooted as soon as possible. The <i>nodeid</i> field is not useful in this case.
<code>CMM_MASTER_DEMOTED</code>	The <i>nodeid</i> represents a previous master node that has been demoted. For more information, see “Administrative Attributes” on page 7 .
<code>CMM_MASTER_ELECTED</code>	The <i>nodeid</i> is that of the newly elected master node. A cluster election has selected a new master and the previous master (if any) quits its role. The new node might have just joined the cluster and there might not have been a previous master.
<code>CMM_MEMBER_JOINED</code>	A peer node has joined the cluster. The <i>nodeid</i> is that of the new peer node.
<code>CMM_MEMBER_LEFT</code>	A peer node has the <code>CMM_OUT_OF_CLUSTER</code> role.

TABLE 6-3 Change Notifications (*Continued*)

Value	Description
CMM_STALE_CLUSTER	The master node sends a <i>membership frame</i> every 4 seconds to inform other nodes of the current state of the cluster. If no frames are received by a node for more than 10 seconds, the CMM on this node notifies the local applications. The CMM_STALE_CLUSTER notification means that, even if the CMM API is available, the returned information from a node might not reflect the current state of the cluster. Operations involving the master, such as a new node joining the cluster, might fail because the master is unreachable. This situation is abnormal and recovery actions must be taken. The <i>nodeid</i> field is not useful in this case. Calls that return the CMM_OK value before this notification return CMM_EAGAIN after it while the cluster is in a stale state.
CMM_VICEMASTER_DEMOTED	The <i>nodeid</i> represents a previous vice-master node that has been demoted. This is only sent if the vice-master node is disqualified.
CMM_VICEMASTER_ELECTED	A new vice-master is elected. The node no longer has its previous role. The previous vice-master (if any) is demoted. The <i>nodeid</i> is that of the newly elected vice-master node.
CMM_VALID_STATE	The state of the cluster is now valid and running correctly. The <i>nodeid</i> field is not useful for this notification.

Notifications During Changes in the Cluster State

There are many scenarios in which the state of a cluster changes and registered applications and services receive notifications of changes in the cluster state.

A change in the state of a single node can cause the states of other nodes to change. For example, if a new master node is elected, the roles of both the new master node and the former master node change. When a scenario involves a change in the state of more than one node, several notifications can be sent. When several notifications are sent, the notifications are sent in the order in which the changes occur. The `nhcmmmd` daemon sends the minimum number of notifications that describe a new cluster situation. Instead of sending a notification for each change of state for each node, the `nhcmmmd` daemon bundles the information into the minimum number of notifications.

If peer nodes are communicating correctly, the same notification is sent to all nodes, regardless of their membership role.

In each of the scenarios described in this section, there are two example peer nodes: node A and node B. The roles of these nodes are shown in [TABLE 6-4](#). The transition from one role to another is represented as (Role_A) → (Role_B).

TABLE 6-4 Description of the Roles of Example Nodes A and B

Node Role	Description
in	A peer node with a role other than the CMM_OUT_OF_CLUSTER role.
master	A node has the CMM_MASTER role.
vice-master	A node has the CMM_VICEMASTER role.
out	A node has the CMM_OUT_OF_CLUSTER role.

For a summary of the membership roles that a node can have, see [“Membership Roles” on page 5](#).

Scenarios of cluster state change and related notifications are described in the following sections:

- [“Cluster Initialization Notifications” on page 36](#).
- [“Vice-Master Removal Notifications” on page 37](#).
- [“Vice-Master Excluded Notification” on page 37](#).
- [“Peer Node Removal Notification” on page 38](#).
- [“Master Node Excluded Notifications” on page 38](#).
- [“Node Other Than Master Excluded Notification” on page 38](#).
- [“Switchover Notifications” on page 39](#).
- [“Failover Notifications” on page 39](#).
- [“Stale Cluster Notification” on page 41](#).
- [“Amnesia” on page 41](#).
- [“Split Brain” on page 41](#).

Cluster Initialization Notifications

When neither node A nor B is currently running the Foundation Services, nodes A and B are out. When node A becomes the master node, a `MASTER_ELECTED` notification is sent to the registered applications and services. At cluster startup, this is the first step in the creation of a cluster. The notification sent for this scenario is shown in [TABLE 6-5](#).

TABLE 6-5 Master is Elected at Cluster Startup

Transition (node A, node B)	Notifications Sent
(out, out) → (master, out)	<code>CMM_MASTER_ELECTED(A)</code>

The following scenario describes the election of a qualified node to the vice-master role at cluster initialization. This takes place in one step, when the `CMM_VICEMASTER_ELECTED` notification is sent. The notification sent for this scenario is shown in [TABLE 6-6](#).

TABLE 6-6 New Node Joins the Cluster and Becomes Vice-Master

Transition (node A, node B)	Notifications Sent
(master, out) → (master, vice-master)	<code>CMM_VICEMASTER_ELECTED (B)</code>

The following scenario describes when a new node joins the cluster. This node does not take the master or vice-master role and could be a diskless node or a dataless node. The notification sent for this scenario is shown in [TABLE 6-7](#). This scenario can occur at cluster initialization or when a new node is added to a running cluster.

TABLE 6-7 New Node Joins the Cluster

Transition (node A, node B)	Notifications Sent
(master, out) → (master, in)	<code>CMM_MEMBER_JOINED (B)</code>

The following scenario describes the situation where a node that is in becomes vice-master. This scenario can occur if a node is in the cluster but does not immediately declare itself as master-eligible. When its eligibility to be a master node or a vice-master node is known, the node is elected vice-master. This scenario can also occur

if a master-eligible node is disqualified. When the node is requalified, the node becomes that vice-master. The notification sent for this scenario is shown in [TABLE 6-8](#).

TABLE 6-8 Node is Elected Vice-Master

Transition (node A, node B)	Notifications Sent
(master, in) → (master, vice-master)	CMM_VICEMASTER_ELECTED (B)

Vice-Master Removal Notifications

Provided that there is a running vice-master node, if the master node stops being master because its role has been removed, there is a failover, as explained in [“Failover Notifications” on page 39](#).

If the vice-master node stops being vice-master due to a failure, or because its role has been removed, there is no backup for the master node and the cluster loses its 2N redundancy.

If the vice-master role is removed because of a failure or by using the `cmm_membership_remove()` function, the notification is shown in [TABLE 6-9](#).

TABLE 6-9 Vice-Master Node Fails or is Removed With the `cmm_membership_remove()` Function

Transition (node A, node B)	Notifications Sent
(master, vice-master) → (master, out)	CMM_VICEMASTER_DEMOTED (B) CMM_MEMBER_LEFT (B)

Vice-Master Excluded Notification

The vice-master node can be disqualified if you use the `cmm_member_setqualif()` function. The notification sent for this scenario is shown in [TABLE 6-10](#).

TABLE 6-10 Vice-Master is Disqualified With the `cmm_member_setqualif()` Function

Transition (node A, node B)	Notifications Sent
(master, vice-master) → (master, in)	CMM_VICEMASTER_DEMOTED (B)

For more information about disqualifying a node by using the `cmm_member_setqualif()` function, see [“Setting the Qualification of a Node” on page 51](#). Care must be taken with the use of the `cmm_member_setqualif()`

function. Do not trigger a failover. For more information, see [“Triggering a Failover by Using the `cmm_member_setqualif\(\)` Function” on page 55](#). See also the `cmm_member_setqualif(3CMM)` man page.

Peer Node Removal Notification

If a peer node other than the master or vice-master loses its role in the cluster, it becomes temporarily out of the cluster. This occurs if you use the `cmm_membership_remove()` function on the peer node. The notification sent in this scenario is shown in [TABLE 6-11](#).

TABLE 6-11 Node Other Than Master or Vice-Master is Removed From Cluster

Transition (node A, node B)	Notifications Sent
(master, in) → (master, out)	CMM_MEMBER_LEFT(B)

Master Node Excluded Notifications

If the master node fails, the node can be excluded from the cluster as described in [“Removing or Excluding a Node” on page 49](#). The notification sent in this scenario is shown in [TABLE 6-12](#)

TABLE 6-12 Master Node is Excluded From Cluster

Transition (node A, node B)	Notifications Sent
(master, vice-master) → (out, master)	CMM_MEMBER_LEFT(A) CMM_MASTER_ELECTED(B)

Node Other Than Master Excluded Notification

If a node other than the master fails it can be excluded from the rest of the cluster as described in [“Removing or Excluding a Node” on page 49](#). The notification sent is shown in [TABLE 6-13](#).

TABLE 6-13 Node Other Than Master is Excluded From Cluster

Transition (node A, node B)	Notifications Sent
(master, in) → (master, out)	CMM_MEMBER_LEFT(B)

The notification sent for this scenario can also be sent for a diskless node.

Switchover Notifications

A switchover is the scheduled transfer of the `CMM_MASTER` role from the master node to the vice-master node. A switchover is not a failure and does not change the qualification level of the master node. A switchover is not a persistent change. A switchover is usually triggered by the cluster administrator for the maintenance of a node. For more information about the maintenance of nodes, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Cluster Administration Guide*.

The notifications sent in the case of a switchover from the master to the vice-master node, triggered by calling the `cmm_mastership_release()` function, are shown in [TABLE 6-14](#).

TABLE 6-14 Switchover Triggered by the `cmm_mastership_release()` Function

Transition (node A, node B)	Notifications Sent
(master, vice-master) → (vice-master, master)	<code>CMM_MASTER_ELECTED</code> (B) <code>CMM_VICEMASTER_ELECTED</code> (A)

For further information and an example that uses the `cmm_mastership_release()` function to trigger a switchover, see [“Triggering a Switchover” on page 51](#).

Failover Notifications

A failover is the unscheduled transfer of the `CMM_MASTER` role from the master node to the vice-master node. A failover is a response to the removal or failure of the master node or disqualification of the master node. This section describes two failover scenarios:

- [“Failover Due to the Removal or Failure of the Master Node” on page 39](#).
- [“Failover Due to Master Disqualification” on page 40](#).

Failover Due to the Removal or Failure of the Master Node

If master node is removed from the cluster by using the `cmm_membership_remove()` function, the node takes `CMM_OUT_OF_CLUSTER` role. This role indicates that the node is out of the cluster, but is configured to be in the cluster, and has access to cluster information. This is described in [“Membership Roles” on page 5](#).

The notification sequence is the same, whether a master failover occurs because the master node fails or because the master role is removed. The master node is excluded from the cluster and the vice-master becomes the master. The notifications sent for this scenario are shown in [TABLE 6-15](#).

TABLE 6-15 Failover Due to the Removal or Failure of the Master Node

Transition (node A, node B)	Notifications Sent
(master, vice-master) → (out, master)	CMM_MASTER_DEMOTED (A) CMM_MEMBER_LEFT (A) CMM_MASTER_ELECTED (B)

The `nhcmmd` daemon issues notifications of this failover, described in [“Introduction to Change Notifications”](#) on page 31.

For an example of how to trigger a failover using the `cmm_membership_remove()` function, see [“failover.c”](#) on page 54.

Failover Due to Master Disqualification

In this scenario, the failover of the master node is due to the use of the `cmm_member_setqualif()` function. The master node is no longer able to be either master or vice-master, but is in the cluster as a peer node. The vice-master becomes the master node. Because there is no other master-eligible node to take the vice-master role, the cluster loses its 2N redundancy. The former master node must be requalified to restore 2N redundancy. The notifications sent for this scenario are shown in [TABLE 6-16](#).

TABLE 6-16 Failover Due to the Disqualification of the Master Node

Transition (node A, node B)	Notifications Sent
(master, vice-master) → (in, master)	CMM_MASTER_DEMOTED (A) CMM_MASTER_ELECTED (B)

The `nhcmmd` issues notifications of this failover, described in [“Introduction to Change Notifications”](#) on page 31.

The `cmm_member_setqualif()` function is used during the process of peer node reboot and is called from the node that is being rebooted by the service coordinating the node reboot.

For an example of how to trigger a failover using the `cmm_member_setqualif()` function, see [“Triggering a Failover by Using the `cmm_member_setqualif\(\)` Function”](#) on page 55.

Stale Cluster Notification

When information received by a peer node from the master node is more than 10 seconds old, the information is considered to be stale. A stale cluster does not guarantee that there is no change in the cluster. A stale cluster means that information held by the master node is not reaching a peer node. This can happen if the master node is not functioning correctly and does not send information to the peer node. A stale cluster can also occur if the master node does send information but it does not reach the peer node, due for example to a problem in the network. The notification sent for this scenario is shown in [TABLE 6-17](#).

TABLE 6-17 Cluster is in a Stale State

Transition (node A, node B)	Notifications Sent
(master, any) → (stale cluster)	CMM_STALE_CLUSTER(0)

Amnesia

Amnesia is an error condition in which a cluster restarts with stale cluster configuration data. This can happen when a cluster is restarted from a node that was not previously part of the most recent cluster membership list.

Split Brain

Split brain is an error condition in which there are two master nodes. This can be caused by interconnect failure between peer nodes.

During split brain, each master node assumes that it is the only master node in the cluster. A split brain can begin with any combination of roles for nodes A and B. The notification sent for this scenario is shown in [TABLE 6-18](#).

TABLE 6-18 Two Masters in the Cluster (Split Brain)

Transition (node A, node B)	Notifications Sent
(any, any) → (master, master)	CMM_INVALID_CLUSTER

For information about how to recover from a split brain error condition, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Troubleshooting Guide*.

Managing Changes in the Cluster State

This chapter describes how to receive and react to notifications about changes in the cluster state, with examples on how to respond to these notifications by modifying the state of the cluster. For more information, see the following topics:

- [“Setting a Timeout Value for Calls to the nhcmmnd Daemon” on page 43](#)
- [“Reloading the Cluster Node Table” on page 45](#)
- [“Receiving and Handling Change Notifications” on page 45](#)
- [“Responding to Cluster Notifications by Modifying the Cluster” on page 49](#)

Setting a Timeout Value for Calls to the nhcmmnd Daemon

The timeout parameter is used globally by the CMM API to signify the maximum amount of time a call can block. A different timeout can be set for each client.

Using the `cmm_connect()` function, you can:

- Call the `nhcmmnd` daemon.
- Set the timeout value for subsequent calls to the `nhcmmnd` daemon. The default value is five seconds.

Note – The `cmm_connect()` function is called implicitly by the first call to the CMM API. You do not need to call the `cmm_connect()` function to create a connection between an application and the `nhcmmnd` daemon on a node. If you do not set the timeout, it remains at the default value of five seconds.

The new value of the timeout is not used by the call with which you set it.

The `cmm_connect()` function can be called from any node, even a node that has been excluded from the cluster for administrative reasons. The `cmm_connect()` function does not use information provided by the CMM API. For further information, see the `cmm_connect(3CMM)` man page.

The `cmm_disconnect()` function closes the connection between the current calling process and the `nhcmmd` daemon. For more information about this, see the `nhcmmd(1M)` man page, or for the Linux OS, refer to the `nhcmmd(8)` man page and `nhfs.conf(4)` man pages. This frees the resources allocated to the client connection. If notifications were registered, they are no longer sent. For information about notifications, see [Chapter 6](#).

The `cmm_connect()` and `cmm_disconnect()` functions cannot be called within a callback function.

The following example demonstrates how to use the `cmm_connect()` function to set a timeout:

CODE EXAMPLE 7-1 `connect.c`

```
#include <stdio.h>
#include <cmm/cmm.h>

#include "common.h"

int main(void)
{
    cmm_error_t result;
    struct timespec time_out = { 1 /*seconds*/,
                                500000000 /* nanoseconds */};

    result
    if (result != CMM_OK) {
        printf("Couldn't set the timeout: %s\n",
            cmm_strerror(result));
        return 1;
    }

    /*
    /* Your code here
    */
    return 0;
}
```

Reloading the Cluster Node Table

The `cmm_config_reload()` function can be called from the master node to make the `nhcmmnd` daemon reload the cluster node table. Use this function when a node is added to or removed from the cluster node table.

The only permitted operations here are addition and removal of a node. For more information about adding and removing a node, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Cluster Administration Guide*.

You cannot use the cluster node table to edit a node's attributes. Use the CMM API to do this. For example, use the `cmm_member_setqualif(3CMM)` function.

For more information, see the `cmm_config_reload(3CMM)`, `cluster_nodes_table(4)`, and `nhcmmnd(1M)`, or for the Linux OS, the `nhcmmnd(8)`, man pages.



Caution – If you want to add a node make sure it is powered on before you reload the cluster node table. If you want to remove a node make sure it is powered off before you reload the cluster node table. This ensures that you cannot remove the master node from the table.

Receiving and Handling Change Notifications

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

You can use the CMM API to register and unregister for notifications that indicate a change in the cluster. You can also configure your applications to filter, receive, and dispatch these notifications.

This information applies to the CMM API only and is separate from the process of registering for notifications sent by the Node Management Agent (NMA).

For more information, see “Registering to Receive Notifications” in the *Netra High Availability Suite 3.0 1/08 Foundation Services NMA Programming Guide*.

You can manage the handling of notifications in general by using the following functions:

- `cmm_cmc_register()`
- `cmm_cmc_unregister()`
- `cmm_cmc_filter()`
- `cmm_notify_getfd()`
- `cmm_notify_dispatch()`

This sections that follow contain these topics:

- [“Registering to Receive Notifications” on page 46.](#)
- [“Filtering Notifications” on page 46.](#)
- [“Receiving and Dispatching Notifications” on page 47.](#)
- [“Retrieving Change Notifications” on page 47.](#)

Registering to Receive Notifications

To receive notifications, the applications or services can use the `cmm_cmc_register()` function to register a callback with the `nhcmmmd` daemon. When a membership change occurs in the cluster, the `nhcmmmd` daemon notifies the application or service through the callback function by sending a notification. Applications or services receive notifications by polling, using a function such as the `poll()` function. For more information, see the `poll(2)` man page.

To change a registration, an application or service must first cancel the existing registration by using the `cmm_cmc_unregister()` function, and then register a new notification by using the `cmm_cmc_register()` function.

For an example of how to use these functions, see [CODE EXAMPLE 7-2](#). For further information, see the `cmm_cmc_register(3CMM)` and `cmm_cmc_unregister(3CMM)` man pages.

Filtering Notifications

By default, when an application registers to receive notifications, the application receives a notification for every change in the cluster state. These notifications can be filtered by using the `cmm_cmc_filter()` function.

For each notification present in the filter, as selected by using the `cmm_cmc_filter()` function, the registered callback is invoked. The defined filter is applied for further calls to the `cmm_notify_dispatch()` function.

For an example of how to use these functions, see [CODE EXAMPLE 7-2](#). For further information about how to use the `cmm_cmc_filter()` function, see the `cmm_cmc_filter(3CMM)` man page.

Receiving and Dispatching Notifications

Applications or services that are registered to receive notifications can use the `cmm_notify_getfd()` function. This function returns the file descriptor through which the notifications are delivered.

An application uses a polling function, such as `poll()`, to monitor the file descriptors returned by `cmm_notify_getfd()`. When the polling function indicates activity on this file descriptor, `cmm_notify_dispatch()` must be called. For each pending notification, before invoking the callback, the CMM API checks that this notification is present in the filter (as selected with `cmm_cmc_filter()`).

The callback is invoked with the notification and the *client_data* argument passed to the `cmm_cmc_register()` function.

For an example of how to use these functions, see [CODE EXAMPLE 7-2](#). For further information see the `cmm_cmc_register(3CMM)`, `cmm_notify_getfd(3CMM)`, `cmm_notify_dispatch(3CMM)`, and `poll(2)` man pages.

Retrieving Change Notifications

The following example demonstrates how to use the functions that enable you to filter, receive and dispatch notifications:

CODE EXAMPLE 7-2 `notif.c`

```
#include <stdio.h>
#include <poll.h>
#include <cmm/cmm.h>

#include "common.h"

static void notif_callback(const cmm_cmc_notification_t *notif,
                          void *data)
{
    cmm_member_t member ;

    switch (notif->cmchange) {
    case CMM_VICEMASTER_ELECTED:
        printf("Node %u elected vicemaster\n", notif->nodeid);
        if (cmm_vicemaster_getinfo(&member) == CMM_OK)
```

CODE EXAMPLE 7-2 notif.c (Continued)

```

        print_member(&member) ;
        break ;
    case CMM_VICEMASTER_DEMOTED:
        printf("vicemaster %u demoted\n", notif->nodeid);
        break ;
    default::
        printf("Unexpected notification received\n",
            break ;
    }
}

int main(void)
{
    cmm_error_t      result;
    cmm_cmchanges_t  wanted_notifs[2] = { CMM_VICEMASTER_ELECTED,
                                           CMM_VICEMASTER_DEMOTED } ;

    struct pollfd    notif_poll;
    int              poll_res;

    result = cmm_cmc_filter(CMM_CM_NOTIFY_SET, wanted_notifs, 2);
    if (result != CMM_OK) {
        printf("Couldn't set notification filter: %s\n",
            cmm_strerror(result)) ;
        return 1;
    }

    result = cmm_notify_getfd(&notif_poll.fd);
    if (result != CMM_OK) {
        printf("Couldn't get notification fd: %s\n",
            cmm_strerror(result));
        return 1;
    }

    notif_poll.events = POLLIN;
    notif_poll.revents = 0;

    fprintf ("Waiting for notifications\n");

    while (notif_poll.revents & POLLHUP) == 0) {

        poll_res = poll(&notif_poll, 1, -1);
        if (poll_res == 1 && (notif_poll.revents & POLLHUP) == 0) {
            result = cmm_notify_dispatch();
            if (result != CMM_OK) {
                printf("Failed dipatch: %s\n",
                    cmm_strerror(result));
                /* Force leaving the loop */

```



```

        notif_poll.revents = POLLHUP
    }
}
(void) cmm_cmc_unregister();
}
return 0;
}

```

Responding to Cluster Notifications by Modifying the Cluster

You can use the CMM API to respond to notifications received from the `nhcmmnd` daemon that indicate a change in the cluster. In response to these notifications, it might be necessary to remove or disqualify a node or to trigger a switchover or a failover.

This sections that follow contain these topics:

- [“Removing or Excluding a Node” on page 49](#)
- [“Setting the Qualification of a Node” on page 51](#)
- [“Triggering a Switchover” on page 51](#)
- [“Triggering a Failover” on page 53](#)

Removing or Excluding a Node

A node can be removed from the cluster by using the `cmm_membership_remove()` function. The `cmm_membership_remove()` function temporarily takes the current node out of the cluster by giving it the `CMM_OUT_OF_CLUSTER` role and all other peer nodes learn that it is not an active peer node.

A node with this role is not actually excluded from the cluster. It is still configured to be in the cluster. For an explanation of this role, see [“Membership Roles” on page 5](#).

If you want to exclude a node completely from the cluster, first use the `cmm_membership_remove()` function to remove the node from the cluster. Then remove the entry for this node from the cluster node table. It is better to remove the node completely from the cluster node table instead of attributing the node with an excluded value (X) in the `cluster_nodes_table` file. For more information, see the `cluster_nodes_table(4)` man page.

The `nhcmmd` issues notifications of the node's exclusion. For more information, see [“Triggering a Failover by Using the `cmm_membership_remove\(\)` Function” on page 53](#).

Removing the Master Node

Removing the master node must be done in two stages so as to avoid triggering a failover. First, the node that is master should be released from the role of master by using the `cmm_mastership_release()` function. This triggers a switchover. Only then should the node be removed from the cluster. The node is removed from the cluster by calling the `cmm_membership_remove()` function from the node. This gives the node the `CMM_OUT_OF_CLUSTER` role, effectively taking the node out of the cluster. For more information about how to trigger a switchover, see [“Triggering a Switchover” on page 51](#). For an explanation of the notifications sent when you trigger a switchover, see [“Switchover Notifications” on page 39](#).

If the `cmm_membership_remove()` function is called directly from the master node, without first triggering a switchover to a qualified vice-master node, then a failover is triggered.



Caution – Triggering a failover should be done for test purposes only.

For more information about how to trigger a failover, see [“Triggering a Failover” on page 53](#).

For an explanation of the notifications sent when you remove the master role with the `cmm_membership_remove()` function, see [TABLE 6-15](#). For an explanation of the notifications sent when you trigger a failover, see [“Failover Notifications” on page 39](#).

Removing the Vice-Master Node

When the `cmm_membership_remove()` function is called by a system service on the vice-master node, the function removes the vice-master node from the cluster. If the vice-master is removed from the cluster, the cluster no longer has 2N redundancy.

You can remove the vice-master node for maintenance purposes. If you want to perform maintenance on *both* master-eligible nodes, remove the vice-master and carry out the maintenance. Then trigger a switchover, remove the new vice-master and perform the maintenance tasks on this node. For more information, see the *Netra High Availability Suite 3.0 1/08 Foundation Services Cluster Administration Guide*.

For an explanation of the notifications sent when you remove the vice-master node with the `cmm_membership_remove()` function, see [TABLE 6-9](#).

Removing Diskless and Dataless Nodes

If the `cmm_membership_remove()` function is called by a system service on a diskless node, the function removes the diskless node from the cluster. This action has no effect on the role of other nodes in the cluster.

Setting the Qualification of a Node

The `cmm_member_setqualif()` function sets the qualification of a node. Qualification level is only relevant for master-eligible nodes. This function can only be called from the master node. The *nodeid* and the qualification level must be provided as input parameters to this function. The *nodeid* of the current node can be retrieved by using the `cmm_node_getid()` function.

The following example demonstrates how to use the `cmm_member_setqualif()` function to disqualify a node with the *nodeid* 12.

CODE EXAMPLE 7-3 Disqualifying a Node

```
new_qualif = (qualify ? CMM_QUALIFIED_MEMBER :
               CMM_DISQUALIFIED_MEMBER);
if (cmm_member_setqualif (12, CMM_DISQUALIFIED_MEMBER) != CMM_OK)
    /* handle the error */;
```

Triggering a Switchover

A switchover is usually triggered by the system administrator for maintenance of a node. There are two ways to trigger a switchover:

- By calling the `cmm_mastership_release()` function.
- By using the `nhcmmstat` tool.

For information about the `nhcmmstat` tool, see the `nhcmmstat(1M)` man page.

Triggering a Switchover Using `cmm_mastership_release()`

The `cmm_mastership_release()` function enables a calling process to trigger a switchover. This function must be called from the master node. If the vice-master node is qualified to be master when the `cmm_mastership_release()` function is called, it becomes the master node. If there is no node qualified to become master when the `cmm_mastership_release()` function is called, the function does not release the mastership from the current master and the function fails.

After the `cmm_mastership_release()` function is called, the calling node remains master until the vice-master node has taken the master role. When this happens, the `nhcmmmd` daemon issues a notification of the switchover. For information about notifications during a switchover, see [“Switchover Notifications” on page 39](#). For more information about the `nhcmmmd` daemon and notifications, see [“Introduction to Change Notifications” on page 31](#).

The `cmm_mastership_release()` call is synchronous, that is, it only returns when the switchover has been completed or has failed, or when a timeout occurs.

CODE EXAMPLE 7-4 demonstrates how to use the `cmm_mastership_release()` function to trigger a switchover:

CODE EXAMPLE 7-4 `switchover.c`

```
#include <stdio.h>
#include <cmm/cmm.h>

int main(void)
{
    cmm_error_t  result;

    result = cmm_mastership_release();
    switch (result) {
    case CMM_OK:
        printf("Switch-over requested\n");
        break;
    case CMM_EPERM:
        printf("Not running on the master node\n");
        break;
    case CMM_ECANCELED:
        printf("No vicemaster in the cluster\n");
        break;
    default:
        printf("Failed to release the mastership: %s\n",
               cmm_strerror(result));
        break;
    }
}
```

```

    return (result == CMM_OK ? 0 : 1);
}

```

Triggering a Failover

A failover can be triggered by removing the master node by using the `cmm_membership_release()` function, or by disqualifying the master node by using the `cmm_member_setqualif()` function.

The sections that follow contain these topics:

- [“Triggering a Failover by Using the `cmm_membership_remove\(\)` Function” on page 53.](#)
- [“Triggering a Failover by Using the `cmm_member_setqualif\(\)` Function” on page 55.](#)



Caution – Trigger a failover only for test purposes.

Triggering a Failover by Using the `cmm_membership_remove()` Function

The `cmm_membership_remove()` function removes from the cluster the node from which the `cmm_membership_remove()` function is called. If you call this function from the master when the vice-master is synchronized, you trigger a failover. When the `cmm_membership_remove()` function is called from the master node, the master node stops sending heartbeat information to other nodes in the cluster, which triggers a failover. The notifications sent for this scenario are shown in [“Failover Due to the Removal or Failure of the Master Node” on page 39.](#)

If there is no vice-master, or the master-eligible nodes are desynchronized, the `CMM_ECANCELLED` error is returned. If a node is not configured to be in any cluster, any subsequent call to the CMM API returns the `CMM_ENOCLUSTER` error. For further information about these and other return values, see [“Return Values of the CMM API” on page 62.](#)

The only way for a removed node to rejoin the cluster is to restart the `nhprobed` daemon and the CMM service.

For further information, see the `cmm_membership_remove(3CMM)` man page and the `nhprobed(1M)`, or for the Linux OS, refer to the `nhprobed(8)`, man page.

An example that demonstrates how to trigger a failover by using the `cmm_membership_remove()` function is shown in [CODE EXAMPLE 7-5](#).

CODE EXAMPLE 7-5 failover.c

```
#include <stdio.h>
#include <cmm/cmm.h>

int main(void)
{
    cmm_error_t result;
    cmm_nodeid_t curr_node;
    cmm_member_t master_info;

    result = cmm_node_getid(&curr_node);
    if (result != CMM_OK) {
        printf("Couldn't get node id: %s\n", cmm_strerror(result));
        return 1;
    }

    result = cmm_master_getinfo(&master_info);
    if (result != CMM_OK) {
        printf("Couldn't get node id: %s\n", cmm_strerror(res));
        return 1;
    }

    if (master_info.nodeid != curr_node) {
        printf("Not running on the master node\n");
        return 1;
    }

    result = cmm_membership_remove();
    switch (result) {
    case CMM_OK:
        printf("Failover triggered and node removed from the
            cluster\n");
        break;
    case CMM_OK:
        printf("Failover triggered and node removed from the
            cluster\n");
        break;
    default:
        printf("Failed to release the membership: %s\n",
            cmm_strerror(result));
        break;
    }
}
```

```

    return (result == CMM_OK ? 0 : 1);
}

```

Triggering a Failover by Using the `cmm_member_setqualif()` Function

A failover can be triggered by calling the `cmm_member_setqualif()` function on the master node. The notifications sent for this scenario are shown in [TABLE 6-16](#).

When the master node fails, if the vice-master node cannot take over the master role, another master-eligible node must be qualified to become master.

The vice-master node cannot take over the master role if, for example, the vice-master node is not synchronized with the master node. If this is the case, and if no other node in the cluster is qualified to become the master, a master-eligible node must be qualified using the `cmm_member_seizequalif()` function. For more information about the `cmm_member_seizequalif()` function, see the `cmm_member_seizequalif(3CMM)` man page.

The `cmm_member_seizequalif()` function is used by the `cmm_member_setqualif` command and the `squalif` command of the `nhcmmstat` tool.

Rejoining the Cluster

When a node has the `CMM_OUT_OF_CLUSTER` role, two of the possible reasons are:

- It is configured not to join the cluster automatically (`CMM.StartUp.Join=False` in `nhfs.conf(4)`).
- It has been removed from the cluster after a call to `cmm_membership_remove()`.

If the node has the `CMM_OUT_OF_CLUSTER` role for one of these two reasons, then you can rejoin it to the cluster using a call to the function `cmm_membership_include()`. If the node is already in the cluster, the function will succeed.

A successful return of this function does not mean that the node has joined the cluster. It means that the request is being taken into consideration and an attempt will be made to rejoin the node to the cluster. The node could fail to join the cluster if some of the required conditions are not met, such as if the node is excluded or disqualified.

CODE EXAMPLE 7-6 join.c

```
#include <stdio.h>
#include <cmm/cmm.h>

int main(void)
{
    cmm_member_t member_info;
    cmm_error_t result;
    cmm_nodeid_t current;

    result = cmm_node_getid(&current);
    if (result != CMM_OK) {
        printf("Failed to retrieve the current node's id\n: %s",
            cmm_strerror(result));
        return 1;
    }

    result = cmm_potential_getinfo(current, &member_info);
    switch (result) {
    case CMM_OK:
        break;
    case CMM_ESRCH:
        printf("Current node is NOT configured in the cluster\n");
        break;
    default:
        printf("Couldn't get member node information: %s\n",
            cmm_strerror(result));
        break;
    }

    if (!cmm_member_isoutofcluster(&member_info)) {
        printf("Node is already in the cluster\n");
    } else {
        result = cmm_membership_include();
        switch (result) {
        case CMM_OK:
            printf("Trying to join the cluster\n");
            cmm_strerror(result));
            break;
        }
    }
}
```


CODE EXAMPLE 7-6 `join.c` (*Continued*)

```
        return (result == CMM_OK ? 0 : 1);  
    }
```


Debugging Applications in the Foundation Services

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

- [“Reporting Application Errors” on page 59](#)
- [“Reading Error Information for Debugging” on page 60](#)
- [“Stopping the Daemon Monitor for Debugging” on page 60](#)
- [“Broken Pipe Error Messages” on page 61](#)
- [“Return Values of the CMM API” on page 62](#)

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 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

The standard return values for CMM API errors are summarized in [TABLE 8-1](#).

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 [Chapter 6-1](#).

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 `nhpmd(1M)`, or for the Linux OS, the `nhpmd(8)`, 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_00000000 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.conf(4)` man page.

Return Values of the CMM API

The CMM API provides extensive return values for errors and successful function calls. They are listed in [TABLE 8-1](#).

TABLE 8-1 Common Return Values of the CMM API

Return Value	Result	Possible Responses
CMM_OK	The function call succeeded.	None required.
CMM_EAGAIN	Returned information is based on a cluster view that has not been updated by the master node for more than 10 seconds.	Retry the function call.
CMM_EBADF	An identifier or descriptor that corresponds to a file descriptor is invalid. The connection to the CMM is no longer valid. Perhaps the CMM is dead.	Verify that data in your program is not corrupted. Call the <code>cmm_cmc_register()</code> and the <code>cmm_notify_getfd()</code> functions to fetch a new connection.
CMM_EBUSY	<ul style="list-style-type: none">For all functions: The CMM API server is temporarily out of resources to respond to the requested operation.For <code>cmm_cmc_unregister()</code>: An attempt to unregister a callback, that is, a call to the <code>cmm_cmc_unregister()</code> function, failed because the caller's callback function is active. See the <code>cmm_cmc_unregister(3CMM)</code> man page.	Wait, then retry the function call. You can decide the length of wait, based on the application's characteristics.
CMM_ECANCELED	A switchover operation was cancelled. For example, when trying to demote the master, no vice-master can take over the master role.	Continue.
CMM_ECONN	The local CMM API process is unreachable.	Check that the process is currently running. Perhaps it is not running yet. Retry the function call.
CMM_EEXIST	Only one function can be registered at a time. An attempt to call the <code>cmm_cmc_register()</code> function when a callback is already registered returns this message.	The calling process has already registered a callback. Verify that the existing function is required for the purpose of your program.

TABLE 8-1 Common Return Values of the CMM API (*Continued*)

Return Value	Result	Possible Responses
CMM_EINVAL	A function parameter has an invalid value.	<p>Ensure that the type of each parameter matches the type in the function prototype. For example the <i>nodeid</i> is not a master-eligible node.</p> <p>Cast variables to the expected type if necessary and verify that the area of memory that stores the parameter is valid.</p>
CMM_ENOCLUSTER	<p>One of the following has occurred:</p> <ul style="list-style-type: none"> • The local node is not configured in an active cluster. This occurs, for example, when the cluster election is in progress. • The local node has been removed from the cluster node table on the master node. For more information, see the <code>cluster_nodes_table(4)</code> man page. • There is more than one master node. • The master node has been disqualified and no vice-master node has taken over the master role. • A failover has been triggered by the disqualification of the master node. During the failover, there is a brief time when there is no master node. The CMM_ENOCLUSTER error was returned during this time. 	<p>Any combination of the following:</p> <ul style="list-style-type: none"> • Add an entry for the node to the cluster node table. • Requalify the node. • Assign only one master.
CMM_ENOENT	An attempted operation on an item failed because the item does not exist. For example, when calling the <code>cmm_cmc_unregister()</code> function, no callback has been registered. Not critical.	<p>Any combination of the following:</p> <ul style="list-style-type: none"> • Verify that the area of memory that stores the item is valid. • If you want to delete the item, continue.
CMM_ENOMSG	An attempt to dispatch an event failed because there are no events to be dispatched.	Continue.
CMM_ENOTSUP	The operation could not be correctly executed. This error can be the result of a system problem such as a file that cannot be created or a problem with Remote Procedure Call (RPC) services.	Examine the system log files.

TABLE 8-1 Common Return Values of the CMM API *(Continued)*

Return Value	Result	Possible Responses
CMM_EPERM	The call tried to execute on a node other than the master node, but it can execute only on the master node. For more information, see the <code>cmm_mastership_release(3CMM)</code> , <code>cmm_member_setqualif(3CMM)</code> , and <code>cmm_member_seizequalif(3CMM)</code> man pages.	Execute the function only on the master node.
CMM_ERANGE	The number of cells in the table is smaller than the number of nodes in the cluster. Returned by the <code>cmm_member_getall()</code> function. See the <code>cmm_member_getall(3CMM)</code> man page.	Add an entry in the table for each potential peer node.
CMM_ESRCH	<ul style="list-style-type: none"> Using the <code>cmm_member_getinfo()</code> function to obtain information about a node that is either not in the local cluster node table, or is in the local cluster node table but currently has the <code>CMM_OUT_OF_CLUSTER</code> role. Using the <code>cmm_potential_getinfo()</code> function to obtain information about a node that is not in the local cluster node table. Using the <code>cmm_vicemaster_getinfo()</code> function while the cluster has no vice-master. 	<p>Any combination of the following:</p> <ul style="list-style-type: none"> Examine why the master-eligible node is down or isolated. Add an entry for this node to the cluster node table. See the <code>cluster_nodes_table(4)</code> man page. Change the node's role to master or vice-master.
CMM_ETIMEOUT	No response even when an operation is retried, until the delay has expired. The function call was timed out.	<p>Any combination of the following:</p> <ul style="list-style-type: none"> Retry the function call. Reduce the load on the system.

Index

Numerics

- 2N redundancy
 - loss of, 37, 50, 55
 - synchronization flag, 7

A

- address space, locking, 15
- administrative attributes, 7, 28, 29
 - CMM_ELIGIBLE_MEMBER, 7, 29
 - CMM_EXCLUDED_MEMBER, 29
 - CMM_FLAG_DISQUALIFIED, 7
 - CMM_FLAG_SYNCHRO_NEEDED, 7
 - CMM_FROZEN_MEMBER, 29
- amnesia, 41
- APIs
 - development environment, 1
 - for managing peer nodes, 1
 - examples of use, 3, 16
- applications
 - connecting to nhcmmnd daemon, 43
 - debugging, 59
 - development environment for, 9
 - disconnecting from nhcmmnd daemon, 44
 - registering for notifications, 47
 - starting automatically, 15
 - testing, 11, 15
- attributes
 - administrative *See* administrative attributes, 7

B

- binary files, location for cluster, 13
- broken pipes, 61

C

- callback functions
 - cluster membership change, 32, 33, 34
 - unregistering failure, 62
- cancelled operations, failures due to, 62
- CFLAGS entry, 14
- cgha_cmm library, 12
- change notifications, 31, 32, 33, 34
- cluster, 15
 - identifying using domainid, 29
 - invalid, 33, 41
 - location of binary files, 13
 - stale, 34, 41
 - state, 31, 34
- Cluster Management Service, programming interface, 1
- cluster node table, 6
 - entries, removing, 49
 - errors, 63, 64
 - reloading, 45
- CMM API
 - changing the state of nodes, 49
 - connection failure, 62
 - examples of use, 2
 - function calls, 2
 - header files, accessing, 3
 - introduction to, 1
 - libraries, accessing, 3
 - multithreading characteristics, 2
 - notifications, 31
 - return values, 62
 - usage, 2

- cmm_cmc_filter function, 32, 46, 47
- cmm_cmc_register function, 32, 46
- cmm_cmc_unregister function, 46
- cmm_cmchanges_t structure, 33
- cmm_config_reload function, 45
- cmm_connect function, 43, 44
- cmm_disconnect function, 44
- CMM_DISQUALIFIED_MEMBER level, 6, 29
- CMM_EAGAIN error, 62
- CMM_EBADF error, 62
- CMM_EBUSY error, 62
- CMM_ECANCELED error, 62
- CMM_ECONN error, 62
- CMM_EEXIST error, 62
- CMM_EINVAL error, 63
- CMM_ELIGIBLE_MEMBER
 - attribute, 7
- CMM_ELIGIBLE_MEMBER attribute, 29
- CMM_ENOCLUSTER error, 63
- CMM_ENOENT error, 63
- CMM_ENOMSG error, 63
- CMM_ENOTSUP error, 63
- CMM_EPERM error, 64
- CMM_ERANGE error, 64
- CMM_ESRCH error, 64
- CMM_ETIMEDOUT error, 64
- CMM_EXCLUDED_MEMBER attribute, 29
- CMM_FLAG_DISQUALIFIED
 - attribute, 7
- CMM_FLAG_SYNCHRO_NEEDED
 - attribute, 7
- CMM_FROZEN_MEMBER attribute, 29
- CMM_INVALID_CLUSTER notification, 33, 41
- CMM_MASTER role, 5, 30
- CMM_MASTER_DEMOTED notification, 33, 40, 55
- CMM_MASTER_ELECTED notification, 33, 55
- cmm_master_getinfo function, 22
- cmm_mastership_release function, 39, 52
- cmm_member_getall function, 25
- cmm_member_getcount function, 25
- cmm_member_getinfo function, 23, 27
- cmm_member_is* functions
 - code example, 16
- cmm_member_isdesynchronized function, 27, 29
- cmm_member_isdisqualified function, 27
- cmm_member_isdisqualified function, 29
- cmm_member_iseligible function, 27, 29
- cmm_member_isexcluded function, 29
- cmm_member_isfrozen function, 29
- cmm_member_ismaster function, 26, 30
- cmm_member_isoutofcluster function, 26, 30
- cmm_member_isqualified function, 27, 29
- cmm_member_isvicemaster function, 26, 30
- CMM_MEMBER_JOINED notification, 33, 36
- CMM_MEMBER_LEFT notification, 33, 49
- cmm_member_seizequalif function, 55
- cmm_member_setqualif function, 37, 51, 55
- cmm_member_t structure
 - fields, 28
 - returned by functions, 23
 - sflag field, 29
 - usage, 28
- cmm_membership_release function, 40
- cmm_membership_remove function, 37, 49, 51, 53
- cmm_node_getid function, 21
- cmm_notify_dispatch function, 46, 47
- cmm_notify_getfd function, 46, 47
- cmm_notify_t structure, 32, 34
- CMM_OK return value, 62
- CMM_OUT_OF_CLUSTER role, 5, 30, 49
- cmm_potential_getinfo function, 23, 27
- CMM_QUALIFIED_MEMBER level, 6, 29
- CMM_STALE_CLUSTER notification, 34, 41
- CMM_VALID_STATE notification, 34
- CMM_VICEMASTER role, 5, 30
- CMM_VICEMASTER_DEMOTED notification, 34, 37
- CMM_VICEMASTER_ELECTED notification, 34, 36, 39
- cmm_vicemaster_getinfo function, 22
- commands, nhcmmrole, 27
- common.h library file, code example, 17
- compiling
 - programs, 14

D

Daemon Monitor, monitoring the `nhcmmnd`
daemon, 12

daemons

- issues when debugging, 60
- `nhcmmnd`, 43, 61
 - sending notifications, 31
- `nhprobed`, 45

data transfer, 10

debugging applications
overview, 59

development environment, 1, 9

development host

- disk space, 10
- requirements, 9
- software requirements, 10
- transferring data, 10

diskless nodes, excluding from cluster, 51

disks, space on development host, 10

display messages, formatting, 28

E

elections

- notifications of, 33, 55
- participation in, 7, 27

eligibility, retrieving information, 27

environment variables, `LD_BIND_NOW`, 15

errors

- checking, 59
- `CMM_EAGAIN`, 62
- `CMM_EBADF`, 62
- `CMM_EBUSY`, 62
- `CMM_ECANCELED`, 62
- `CMM_ECONN`, 62
- `CMM_EEXIST`, 62
- `CMM_EINVAL`, 63
- `CMM_ENOCLUSTER`, 63
- `CMM_ENOENT`, 63
- `CMM_ENOMSG`, 63
- `CMM_ENOTSUP`, 63
- `CMM_EPERM`, 64
- `CMM_ERANGE`, 64
- `CMM_ESRCH`, 64
- `CMM_ETIMEOUT`, 64
- displaying, 16
- from function calls, 59
- log files, 60

logging, 59

reporting by the NMA, 60

return values, 60, 62

event, nonexistent, 63

example source code, 3, 16

examples

- location in source code, 3
- using the CMM API, 24

F

failover, 39, 40, 55

caution about triggering, 53

provision for critical applications, 2

qualification levels, 55

failures

- amnesia, 41
- callback functions, 62
- due to cancelled operations, 62
- node, 37
- of the active node, 1
- provision of a standby node, 1
- return values, 62
- split brain, 41
- unregistering callbacks, 62

file descriptors

- invalid, 62
- shortage of, 61

file systems, mounting, 15

files

- copying, 15
- transferring, 15

flags, compiler *See* Makefiles

frozen nodes, identifying, 29

`ftp` command, 14

function calls

- return values, 60
- successful, 62

H

hardware requirements, 9

header files

- in `SUNWnhcmmndpackage`, 11
- locating, 14

I

in nodes, defined, 25

- installation server, transferring data, 10
- invalid cluster, 33, 41
- invalid file descriptors, 62

L

- LD_BIND_NOW environment variable, 15
- LD_FLAGS entry, 14
- libcgha_cmm.so library, 12
 - linking to your application, 12
 - location, 12
- libraries
 - cgha_cmm, 12
 - common.h library file
 - code example, 17
 - installing, overview, 11
 - libcgha_cmm.so, 12
 - linking to your application, 12
 - locating, 14
- linking programs, 14
- locking address space, 15

M

- Makefiles
 - C_FLAGS entry, 14
 - LD_FLAGS entry, 14
- master node
 - definition, 5
 - demotion, 33, 40, 55
 - exclusion from cluster, 50
 - identifying, 26, 30
 - notification of election, 33, 55
 - qualification level, 6
 - reloading cluster node table, 45
 - retrieving information, 22
 - state information, 31
- master-eligible nodes
 - See also* master node
 - See also* qualification levels
 - See also* vice-master node
 - identifying, 29
- membership roles, 28, 29
 - change notifications, 31, 32, 33, 34
 - CMM_MASTER, 5
 - CMM_MASTER, 30
 - CMM_OUT_OF_CLUSTER, 5
 - CMM_OUT_OF_CLUSTER, 30, 49
 - CMM_VICE-MASTER, 5

- CMM_VICEMASTER, 30
 - displaying, 16
 - identifying at command line, 27
 - managing, 2
 - retrieving information about, 6, 26
- memory, locking shared objects in, 15
- messages, formatting, 28
- mlockall function, 15
- mount command, 14
- multithreading characteristics of CMM API, 2

N

- nhcmmmd daemon
 - communicating with libcgha_cmm.so library, 12
 - connecting to applications, 43
 - disconnecting from applications, 44
 - notification pipes, 61
 - sending notifications, 31
 - setting timeout for calls, 43
 - state changes, 49
- nhcmmrole command, 27
- nhcmmstat tool, 51
- nhprobed daemon, 45
- nodes
 - adding, 45
 - addr field, 28
 - administrative attributes, 7, 28, 29
 - availability in cluster, 2
 - CGTP address, 28
 - changing state, 49
 - diskless
 - exclusion from cluster, 51
 - improving performance, 15
 - domainid field, 29
 - eligibility, 27
 - eligibility to become master, 27
 - exclusion from cluster, 29, 38, 49
 - failover, 39, 40
 - failure, 37
 - failures, 1, 37
 - frozen nodes
 - identifying, 29
 - gathering information, 2
 - identifying, 21, 28
 - identifying properties, 27
 - identifying roles, 26, 27
 - in nodes, 25

- incarnation_number field, 28
- information about, 23, 27, 28
- joining the cluster, 31, 33, 36
- last reboot, 28
- leaving the cluster, 31, 37, 49
- master, 5, 22
- master-eligible
 - identifying, 29
- membership of cluster, 7
- membership roles, 5, 28, 29
- name field, 28
- nodeid field, 21, 28
- out nodes, 5
 - identifying, 26, 30
- qualification levels, 6, 27, 29, 51
- removing, 45
- removing role, 37
- retrieving information, 23, 25, 27, 28
- roles, 5
- sflag field, 28, 29
- software_load_id field, 29
- standby, 1
- state change notifications, 34
- state information, 31
- switchover, 39, 52
- synchronization, 7, 27, 29
- vice-master, 5, 22
- nonexistent items, 63
- notifications
 - accessing, 45, 49
 - broken pipes, 61
 - callback functions, 32, 33, 34
 - change, 31, 32, 33, 34
 - CMM_INVALID_CLUSTER, 33, 41
 - CMM_MASTER_DEMOTED, 33, 40, 55
 - CMM_MASTER_ELECTED, 33, 55
 - CMM_MEMBER_JOINED, 33, 36
 - CMM_MEMBER_LEFT, 33, 49
 - CMM_STALE_CLUSTER, 34, 41
 - CMM_VALID_STATE, 34
 - CMM_VICEMASTER_DEMOTED, 34, 37
 - CMM_VICEMASTER_ELECTED, 34, 36, 39
 - code example, 47
 - dispatching, 47
 - filtering, 46
 - messages, 33
 - of state changes, 34
 - receiving, 47
 - registering for, 45, 46, 49

- sent by nhcmmnd daemon, 31
- unregistering for, 45, 49

O

- out nodes, defined, 5

P

- packages
 - prerequisites for CMM API, 3
 - SUNWnhcmmnd, 11
- parameters, invalid, 63
- peer nodes, retrieving information, 27
- performance, enhancing, 15
- pipes, broken, 61
- poll function, 47
- programs
 - compiling, 14
 - enhancing performance, 15
 - linking, 14

Q

- qualification levels, 6, 29, 51
 - CMM_DISQUALIFIED_MEMBER, 6
 - CMM_DISQUALIFIED_MEMBER, 29
 - CMM_QUALIFIED_MEMBER, 6
 - CMM_QUALIFIED_MEMBER, 29
- failover, 55
- location stored, 27

R

- rcp command, 14
- Reliable NFS, synchronization flag, 7
- return values, 60, 62
 - CMM_EAGAIN, 62
 - CMM_EBADF, 62
 - CMM_EBUSY, 62
 - CMM_ECANCELED, 62
 - CMM_ECONN, 62
 - CMM_EEXIST, 62
 - CMM_EINVAL, 63
 - CMM_ENOCLUSTER, 63
 - CMM_ENOENT, 63
 - CMM_ENOMSG, 63
 - CMM_ENOTSUP, 63
 - CMM_EPERM, 64
 - CMM_ERANGE, 64
 - CMM_ESRCH, 64

CMM_ETIMEDOUT, 64
CMM_OK, 62

S

- server out of resources, 62
- shared objects, locking in memory, 15
- software requirements, development host, 10
- source code, examples, 3, 16
- space requirements, of development host, 10
- split brain, 41
- stale cluster, 34, 41
- startup scripts, 15
- state changes
 - notifications, 34
 - overview, 49
- Sun WorkShop TeamWare, 14
- SUNWnhcmd package, 3
 - contents, 3, 16
- SUNWnhcmdpackage, 11
- SUNWnhhad package, 3
- switchover, 39, 52
- synchronization, 7, 27, 29

T

- timeouts
 - responding to, 64
 - setting for nhcmmmd daemon, 43
- transfer of data, 10

V

- vice-master node
 - definition, 5
 - demotion, 34, 37
 - election, 34, 36, 39
 - exclusion from cluster, 50
 - identifying, 26, 30