Netra High Availability Suite Foundation Services 2.1 6/03 CMM Programming Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817–1770–11
September 2004

Adobe PostScript

040812@9495

Contents

Tables

Figures

Examples

Preface

This book describes how to write applications that use the Cluster Membership Manager API for the Netra™ High Availability (HA) Suite Foundation Services 2.1 6/03.

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 the Foundation Services, you must have experience 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 is an advantage.

Before reading this book, read the *Netra High Availability Suite Foundation Services 2.1 6/03 Overview*.

How This Book Is Organized

This book is divided into the following parts and chapters:

- Part I contains 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.

- Part II contains the following chapters:

  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.

- Part III contains the following chapters:

  Chapter 5 describes how to identify and retrieve data about nodes in the cluster.

  Chapter 6 describes the `nhcmmd` 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.

- Appendix A contains the code examples provided with the Foundation Services product.

Related Books

You will require some of the following books from the Foundation Services documentation set:

- *Netra High Availability Suite Foundation Services 2.1 6/03 Overview*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Glossary*
- *What's New in Netra High Availability Suite Foundation Services 2.1 6/03*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Quick Start Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Hardware Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Custom Installation Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Cluster Administration Guide*

- *Netra High Availability Suite Foundation Services 2.1 6/03 Troubleshooting Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 CMM Programming Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 NMA Programming Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Reference Manual*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Standalone CGTP Guide*
- *Netra High Availability Suite Foundation Services 2.1 6/03 Release Notes*
- *Netra High Availability Suite Foundation Services 2.1 6/03 README*

Accessing Sun Documentation Online

The docs.sun.com^{SM} Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Ordering Sun Documentation

Sun Microsystems offers select product documentation in print form. For a list of documents and how to order them, see "Buy printed documentation" at `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`**
		`Password:`

AaBbCc123	Command-line placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. These are called *class* options. Do *not* save the file. (Emphasis sometimes appears in bold online.)

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

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

PART I

Introduction to the CMM API

This part introduces the characteristics, features, and functions of the CMM API, described in the following chapters:

- Chapter 1 provides an overview of the characteristics of the CMM API.
- Chapter 2 the membership roles, qualification levels, and administrative attributes of nodes.

CHAPTER **1**

The Foundation Services CMM API

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

- "Overview" on page 19
- "Characteristics of the CMM API " on page 20

Overview

The CMM API provides a programming interface to the Cluster Membership Manager (CMM). For information about the CMM, see the *Netra High Availability Suite Foundation Services 2.1 6/03 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 Foundation Services is illustrated in Figure 1–1.

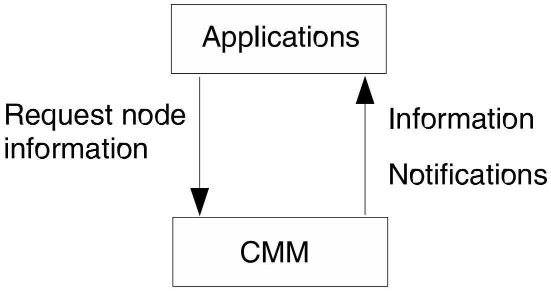

FIGURE 1–1 Interaction of the CMM With Your Applications Using the CMM API

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 `SUNWnhcmd` package. These examples are available after installation on a peer node, in the `/opt/SUNWcgha/examples/cmm_API` directory. For more information, see "CMM API Code Examples" on page 93.

The CMM API header file, `cmm.h`, is located in the `/opt/SUNWcgha/include` directory.

The default location of the CMM API library files in the `/opt/SUNWcgha/lib` directory.

To access the CMM API header file and libraries, the `SUNWnhcmd` package must be installed in your development environment. See Chapter 3 and Chapter 4 for information about creating a development environment for the Foundation Services product.

CHAPTER **2**

Introduction to the Properties of a Node

This chapter introduces the roles, qualification levels and attributes of nodes. For information about the types of nodes, see the *Netra High Availability Suite Foundation Services 2.1 6/03 Overview*.

- "Membership Roles" on page 23
- "Qualification Levels" on page 24
- "Administrative Attributes" on page 25

Membership Roles

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

Master node	A node with membership role `CMM_MASTER`. 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 `CMM_VICEMASTER`. The vice-master node can take over the master role of the cluster. The vice-master node has its own copy of the `cluster_nodes_table` file. For more information, see the `cluster_nodes_table`(4) man page.
Out node	Any node configured to be in the cluster that has the membership role `CMM_OUT_OF_CLUSTER`. 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 "Cluster Model" in the *Netra High Availability Suite Foundation Services 2.1 6/03 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 54.

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:

`CMM_QUALIFIED_MEMBER`	The node is qualified to be master. Only meaningful for eligible nodes.
`CMM_DISQUALIFIED_MEMBER`	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 83.

Administrative Attributes

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

`CMM_ELIGIBLE_MEMBER`
: 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.

`CMM_FLAG_DISQUALIFIED`
: 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.

`CMM_FLAG_SYNCHRO_NEEDED`
: 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 `CMM_FLAG_SYNCHRO_NEEDED` 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.

For more information about Reliable NFS, see the *Netra High Availability Suite Foundation Services 2.1 6/03 Overview*.

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 65. 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 54.

PART II

Setting Up the Development Environment

This part describes how to set up your development host so that you can develop and build applications that use the CMM API. The development environment is described in the following chapters:

- Chapter 3 describes the software requirements of a development host.
- Chapter 4 describes the compiler, Makefile and library requirements you must satisfy before you begin writing applications.

CHAPTER 3

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 29
- "Setting up the Development Host" on page 30
- "Setting up a Foundation Services Cluster" on page 31

For general information about developing applications on the Solaris operating system, see the *Solaris 8 Software Developer Collection*.

Introduction to the Development Environment

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

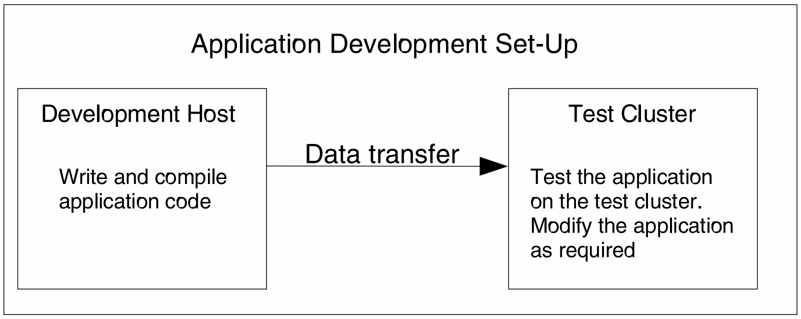
Application Development Set-Up
Development Host
Write and compile application code
Data transfer
Test Cluster
Test the application on the test cluster. Modify the application as required

FIGURE 3–1 Setting up the Development Environment

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

Setting up the Development Host

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

Your development host must have the following software:

Solaris 8 2/02 or above
  The Solaris operating system is the recommended operating system for the Foundation Services.

Forte™ Developer 6 Software Suite (FD6u1), or above
  You can use this software to compile and debug your HA-aware applications.

You can also install Sun WorkShop™ TeamWare 6 update 2 on your development host to manage and configure versions of your code.

Setting up a Foundation Services Cluster

To run applications, you must build them on your development host and deploy them on a cluster that runs the Foundation Services. For information about how to set up a build server and on how to set up a cluster that runs the Foundation Services, see the *Netra High Availability Suite Foundation Services 2.1 6/03 Hardware Guide*.

To install, compile, and run your applications, see Chapter 4.

CHAPTER **4**

Building CMM Applications

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

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 from 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.

For definitions of the terms diskfull, diskless, and dataless nodes, see "Cluster Model" in the *Netra High Availability Suite Foundation Services 2.1 6/03 Overview*.

Installing Libraries and Header Files

Ensure that the library contents and header files of the Foundation Services are available in the runtime environment of your development host. This includes the CMM header (`.h`) files and library (`.so` and `.a`) files. These files are delivered in the `SUNWnhcmd` developer package. In this way these files can be installed on your development host without having to install a running Foundation Services CMM. The `SUNWnhcmd` developer package requires that the `SUNWnhcdt` trace 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 Package and Trace Package

To install the developer package, follow this procedure:

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

```
# pkgadd -d /software-distribution-dir/Packages/ SUNWnhcmd SUNWnhcdt
```

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

After installing the packages, the `libcgha_cmm` library is available in the `/opt/SUNWcgha/lib/` directory, and the header files are available in the `/opt/SUNWcgha/include/` directory.

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 `nhcmmd` daemon, which is monitored by the Daemon Monitor, `nhpmd`. For further information on the `nhcmmd` daemon, see the `nhcmmd`(1M) man page. For more information on the Daemon Monitor, see the `nhpmd`(1M) man page.

You must link the `libcgha_cmm` library to your application. To do this, set the `LD_LIBRARY_PATH` variable to the `/opt/SUNWcgha/lib/` directory.

The services that these libraries use are only available when you run your applications on the fully installed cluster running the Foundation Services. For more information see "Setting up a Foundation Services Cluster" on page 31.

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:

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

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 34.

For an example Makefile that uses a specified code example, see Example 4–4. An example Makefile is provided within the developer package of the Foundation Services, and is shown in "Example Makefile" on page 93.

Compiling Applications

The applications you develop using the CMM API can be compiled using the Sun Forte Developer 6 Software Suite compiler. For more information, refer to the documentation that is supplied with Sun Forte Developer 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) Solaris man page.

If you require fast performance from a program, ensure that shared objects linked with the program are locked in memory at runtime. To lock shared objects in memory at runtime, set the `LD_BIND_NOW` environment variable. For more information on this variable, see the Solaris documentation on "Runtime Linker" in the *Linker and Libraries Guide*.

For better performance from programs running on diskless nodes in a test cluster, you can set the `mlockall()` function within your program to lock address space. For more information, see the `mlockall`(3C) Solaris 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 Foundation Services 2.1 6/03 Hardware Guide*.

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

`ftp`	The `ftp` command is the user interface to the Internet standard File Transfer Protocol. For more information, see the `ftp`(1) man page.
`rcp`	The `rcp` command is used to copy files between machines. For more information, see the `rcp`(1) man page.
`mount`	The `mount` command is used to mount file systems and remote resources. For more information, see the `mount`(1M) man page.

Application Examples

The code fragment shown in Example 4–1 enables the API to display peer node membership. This code fragment should be included when you run the CMM API examples provided in this guide. This code fragment includes `cmm_member_is()` functions, which are explained in Chapter 5.

EXAMPLE 4–1 Mandatory Code Fragment: `common.c`

```
void print_member(
    cmm_member_t const *P_Member)
{
    char L_strRole[16];
    char L_strQualif[16];

    if (cmm_member_ismaster(P_Member))
        strcpy(L_strRole, "MASTER");
    else if (cmm_member_isvicemaster(P_Member))
        strcpy(L_strRole, "VICE-MASTER");
    else if (cmm_member_isoutofcluster(P_Member))
        strcpy(L_strRole, "OUT");
    else
        strcpy(L_strRole, "IN");


    puts("------------------------------") ;
    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", L_strRole) ;
    printf("disqualified   = %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))?"YES":"NO") ;
    printf("excluded    = %s\n",
        (cmm_member_isexcluded(P_Member))?"YES":"NO") ;
    printf("eligible    = %s\n",
        (cmm_member_iseligible(P_Member))?"YES":"NO") ;
    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) ;
    puts("------------------------------") ;
}
```

The code fragment in 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 Example 4–2.

EXAMPLE 4–2 The `common.h` Header File

```
#ifndef __CMM_COMMON__
#define __CMM_COMMON__

#include <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 34.

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, `test_master.c`, is provided. This example uses:

- The `common.h` header file in Example 4–2
- The `print_member()` function from the `common.c` code fragment in Example 4–1

This worked example is shown in Example 4–3.

EXAMPLE 4–3 Example `test_master.c` Program

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

#include "common.h"


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

    code = cmm_master_getinfo(&member_info);
    if (code != CMM_OK) {
        printf("Could not get master info: %s\n", cmm_strerror(code));
        exit(1);
    }

    print_member(&member_info);
    exit(0);
}
```

An example Makefile is also provided in this section. This example Makefile enables you to run the `test_master.c` code in Example 4–3, using:

- The `common.h` header file in Example 4–2
- The code fragment, `common.c` in Example 4–1

This example Makefile is shown in Example 4–4.

EXAMPLE 4–4 Makefile for the `test_master.c` Program

```
CFLAGS = -I/opt/SUNWcgha/include
LDFLAGS = -L/opt/SUNWcgha/lib \
          -lrt -lcgha_cmm

all: master_test

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

common.o: common.c common.h
        $(CC) -c $(CFLAGS) common.c

master_test: master_test.o common.o
        $(CC) $(LDFLAGS) -o master_test master_test.o common.o
```

For more information on setting up a Makefile, see "Setting up a Makefile" on page 35.

The Foundation Services is supplied with source code examples in the `SUNWnhcmd` developer package. These examples are installed in subdirectories of the `/opt/SUNWcgha/examples/` directory. For more information, see "CMM API Code Examples" on page 93.

PART III

Programming By Using the CMM API

For information about how to develop programs with the CMM API, see the following chapters:

- Chapter 5 explains how to use the functions of the CMM API to identify a particular node, its roles and properties, and how to retrieve information about a node in the cluster.
- Chapter 6 describes cluster notifications that indicate changes in the membership or state of the cluster. Notifications or groups of notifications emitted during cluster events are explained in the context of the events.
- Chapter 7 explains how to register for, receive, and filter notifications. This chapter provides examples on how to respond to notifications by modifying the cluster.
- Chapter 8 describes how to handle and report errors in your applications using the CMM API, and how to stop daemons being monitored so that you can debug. This chapter also describes the CMM API error messages and return values.

CHAPTER **5**

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 43
- "Retrieving Information About the Master Node or Vice-Master Node" on page 44
- "Retrieving Information About Any Node" on page 46
- "Retrieving Information About All Nodes in the Cluster" on page 48
- "Identifying the Role of a Node" on page 50
- "Identifying the Properties of a Node" on page 51
- "Using the `cmm_member_t` Structure for Information About Member Nodes" on page 53

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:

EXAMPLE 5–1 Retrieving the *nodeid* of the Current Node

```
#include <cmm.h>
#include <stdio.h>
#include <stdlib.h>         /* for exit() */

/*******************************************************************/
void main(void)
```

EXAMPLE 5–1 Retrieving the *nodeid* of the Current Node *(Continued)*

```
{
    cmm_error_t  res;
    cmm_nodeid_t currnode;

    /* get the current node id */
    if ((res = cmm_node_getid(&currnode))==CMM_OK)
        printf("Current node id is: %d\n", currnode);
    else
        printf("Error getting info on local node: %s\n",
            cmm_strerror(res));
}
```

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 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 disqualification of the master node, 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 89.

The following example tests for the success of a call to the `cmm_master_getinfo()` function:

EXAMPLE 5–2 Testing the Success of the `cmm_master_getinfo()` Function

```
#include <cmm.h>
...
cmm_error_t res;
cmm_member_t member;
...
res = cmm_master_getinfo(&member);
if (res != CMM_OK) {
    /* Handle error. */
    ...
}
...
```

For further information, see the `cmm_master_getinfo`(3CMM) man page.

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 53.

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

EXAMPLE 5–3 Determining Which Node is the Vice-Master

```
#include <cmm.h>
#include <stdio.h>
#include <stdlib.h>        /* for exit() */
#include <strings.h>        /* for strcpy */
#include "common.h"



/******************************************************************/
void main(void)
{
    cmm_error_t  res;
    cmm_member_t vicemaster_info;
    timespec_t   time_out = { 0, 500000 /* musec */};

    }

    /* get info on vicemaster */
    res = cmm_vicemaster_getinfo(&vicemaster_info);
    switch(res) {
    case CMM_ESRCH:
        puts("No Vice master in current cluster");
        break ;
    case CMM_OK:
        puts("Vice master in current cluster is");
        print_member(&vicemaster_info);
        break ;
    default:
        printf("Error getting info on vicemaster: %s\n",
            cmm_strerror(res));
        exit(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 51. 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 53.

If the `cmm_member_getinfo()` function returns the `CMM_ESRCH` error, the node has the `CMM_OUT_OF_CLUSTER` membership role. This error information can also be obtained by using the `cmm_potential_getinfo()` function. For further information about return values, see "Return Values of the CMM API" on page 89.

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:

EXAMPLE 5–4 Retrieving Information About the Current Node Using the `cmm_member_getinfo()` Function

```
#include <cmm.h>
#include <stdio.h>
#include <stdlib.h>         /* for exit() */
#include "common.h"

/*******************************************************************/
void main(void)
{
    cmm_error_t  res;
    cmm_nodeid_t currnode;
    cmm_member_t currnode_info;

    /* get the current node id */
    if ((res = cmm_node_getid(&currnode))==CMM_OK)
        printf("Current node id is: %d\n", currnode);
    else {
        printf("Error getting id of local node: %s\n",
            cmm_strerror(res));
        exit(1) ;
    }
    /* get the current node info */
    res = cmm_member_getinfo(currnode, &currnode_info);
    switch(res) {
    case CMM_OK:
        printf("Current node is in cluster\n");
        break;
    case CMM_ESRCH:
        printf("Current node is *NOT* in any cluster\n");
```

EXAMPLE 5–4 Retrieving Information About the Current Node Using the `cmm_member_getinfo()` Function *(Continued)*

```
        break;
    default:
        printf("Error getting info on local node: %s\n",
            cmm_strerror(res));
        break;
    }
}
```

The following code sample demonstrates how to retrieve information about a specific node by using the `cmm_member_getinfo()` function:

EXAMPLE 5–5 Retrieving Information About a Specific Node in the Cluster Using the `cmm_member_getinfo()` Function

```
#include <cmm.h>
#include <stdio.h>
#include <sys/types.h>       /* for boolean_t */
#include <stdlib.h>          /* for exit(), atoi() */
#include "common.h"

/**************************************************************/
int get_id(cmm_nodeid_t *P_NodeId)
{
    int          success;
    char         str_node[10];
    boolean_t    go_on = B_TRUE;

    while (go_on == B_TRUE) {
        printf("Enter the node id
        of the node you want information about [0 for abort]: ");
        success = (scanf("%9s", str_node) != 0);
        if (success) {
            *P_NodeId = atoi(str_node);
            if (*P_NodeId >= 0)
                go_on = FALSE;
        }
    }
    return (*P_NodeId==0)?B_FALSE:B_TRUE;
}

/**************************************************************/
void main(void)
{
    cmm_error_t  res;
    cmm_nodeid_t onenode;
    cmm_member_t onenode_info;


    if (!get_id(&onenode))
    {
        printf("abort\n");
```

EXAMPLE 5–5 Retrieving Information About a Specific Node in the Cluster Using the `cmm_member_getinfo()` Function *(Continued)*

```
        exit(0);
    }

    /* get the node info */
    res = cmm_member_getinfo(onenode, &onenode_info);
    switch(res) {
    case CMM_OK:
        printf("node %d is in cluster\n", onenode);
        break;

    case CMM_ESRCH:
        printf("node %d is *NOT* in cluster\n", onenode);
        res = cmm_potential_getinfo(onenode, &onenode_info);
        if (res==CMM_OK)
            printf("node %d is member of the cluster\n",onenode);
        else
        if (res==CMM_ESRCH)
            printf("node %d is *NOT* member of the cluster\n",
                                onenode);
        else printf("Error getting info on node %d: %s",
                        onenode,cmm_strerror(res));
        break;

    default:
        printf("Error getting info on node %d: %s\n",
            onenode,
            cmm_strerror(res));
        break;
    }
}
```

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 a table with the information returned in the `cmm_member_t` structure.

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.

EXAMPLE 5–6 Retrieving Information About All Nodes in the Cluster

```
#include <cmm.h>
#include <stdio.h>
#include <stdlib.h>        /* for exit() */
#include <strings.h>        /* for strcpy */
#include "common.h"


/***********************************************************/
void main(void)
{
    cmm_error_t  res;
    cmm_member_t *member_table;
    uint32_t     member_count ;
    uint32_t     index;
    uint32_t     totalitems;

    res = cmm_member_getcount(&totalitems);
    if (res != CMM_OK) {
        fprintf(stderr,
        "Failed to count nodes: error %s\n",
        cmm_strerror(res));
       exit(1) ;
        }

member_table = (cmm_member_t *) malloc (totalitems * sizeof(cmm_member_t));

    if (member_table == NULL) {
        printf("Failed to allocate memory for data -> abort\n");
        exit(1);
    }

    res = cmm_member_getall(totalitems,member_table,&member_count)
    if (res != CMM_OK) {
            fprintf(stderr,
            "Failed to get all nodes: error %s\n",
            cmm_strerror(res));
        free(member_table);
        exit(1) ;
    }

    for (index=0 ; index<member_count ; index++)
        print_member(&(member_table[index])) ;

    free(member_table) ;
}
```

Following is an example output for a two node cluster:

```
------------------------------
node_id     = 12
domain_id   = 28
```

EXAMPLE 5–6 Retrieving Information About All Nodes in the Cluster *(Continued)*

```
name        = one_node
role        = MASTER
qualified   = YES
synchro.    = READY
frozen      = NO
excluded    = NO
eligible    = YES
incarn.     = 998590675
swload_id   = 1
CGTP @      = 10.28.13.12
------------------------------
node_id     = 14
domain_id   = 28
name        = another_node
role        = VICE-MASTER
qualified   = YES
synchro.    = READY
frozen      = NO
excluded    = NO
eligible    = YES
Incarn.     = 998923673
swload id   = 1
CGTP @      = 10.28.13.14
------------------------------
```

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 23.

To find the *nodeid* of a node, see Example 5–1.

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

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

`cmm_member_isoutofcluster`(3CMM) 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 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.

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 54. To get node information, it is better to use the CMM API. Do not to 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 24. The following functions retrieve information about the eligibility and qualification level of a node:

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

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 Example 7–4.

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 25. For more information about the `cmm_member_isdesynchronized()` function, see the `cmm_member_isdesynchronized`(3CMM) man page.

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 might not yet have entered the cluster.

The `cmm_member_getinfo()` and `cmm_potential_getinfo()` functions retrieve information about a node identified by *nodeid*. To find the *nodeid*, see 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.

Example 5–7 demonstrates how to obtain information about the eligibility of the current node to become master. This example uses the following functions:

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

EXAMPLE 5–7 Determining Whether a Node Can Become Master

```
#include <cmm.h>
#include <stdio.h>
#include <stdlib.h>         /* for exit() */
#include "common.h"

/***************************************************************/
void main(void)
{
    cmm_error_t  res;
    cmm_nodeid_t currnode;
    cmm_member_t currnode_info;

    /* get the current node id */
    if ((res = cmm_node_getid(&currnode))==CMM_OK)
        printf("Current node id is: %d\n", currnode);
    else {
        printf("Error getting info on local node: %s\n",
            cmm_strerror(res));
        exit(1);
    }
    /* Get the  node info */
    if ((res = cmm_potential_getinfo(currnode, &currnode_info))!=CMM_OK) {
        printf("Failed to get info on current node -> abort") ;
        exit(1);
```

EXAMPLE 5–7 Determining Whether a Node Can Become Master *(Continued)*

```
    }

    if (!cmm_member_iseligible(&currnode_info)) {
        puts("Local node cannot be master (it is not eligible)");
        exit(0) ;
    }

    if (cmm_member_isdisqualified(&currnode_info)) {
        puts("Local node cannot be master (it is not qualified)");
        exit(0) ;
    }

    if (!cmm_member_isdesynchronized(&currnode_info)) {
        puts("Local node can be master");
        exit(0) ;
    }

    /* Here we know the current node is eligible, not disqualified,
     * desynchronized */
    if (!cmm_member_ismaster(&currnode_info))
        puts("Local node can be Vice-master only");
    else
        puts("Local node can be master (and it is)");

}
```

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:

nodeid	The unique identifier of a node.
name	The user-visible string that identifies a node and is used to format display messages.
addr	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.

incarnation_number	The instant of the last reboot expressed as the number of seconds elapsed since 00:00:00 UTC, January 1, 1970. The *incarnation_number* 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.
sflag	State information about the node. This is a concatenation of the administrative attributes, membership role, and qualification levels. For more detailed information about the `sflag` field, see "Using the `sflag` Field of the `cmm_member_t` Structure" on page 54.
domainid	The unique ID of the cluster that the node can join or has joined. All peer nodes in a cluster have the same *domainid*. A node can belong to only one cluster and the ID of this cluster is the cluster *domainid* of the current node.
software_load_id	This field is set at `1`.

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 53, 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:

`cmm_member_isdesynchronized()`	Determines whether a master-eligible node is desynchronized. While the node has the `CMM_OUT_OF_CLUSTER` role, the qualification level and `CMM_FLAG_SYNCHRO_NEEDED` flag are meaningless.
`cmm_member_isdisqualified()`	Determines whether a master-eligible node has the `CMM_DISQUALIFIED_MEMBER` qualification level. While the node has the `CMM_OUT_OF_CLUSTER` role, the qualification level and `CMM_FLAG_SYNCHRO_NEEDED` flag are meaningless.
`cmm_member_isqualified ()`	Determines whether a master-eligible node has the `CMM_QUALIFIED_MEMBER` qualification level. While the node has the

	`CMM_OUT_OF_CLUSTER` role, the qualification level and `CMM_FLAG_SYNCHRO_NEEDED` flag are meaningless.
`cmm_member_iseligible ()`	Determines whether a node has the `CMM_ELIGIBLE_MEMBER` attribute.
`cmm_member_isexcluded()`	Determines whether a node has the `CMM_EXCLUDED_MEMBER` attribute.
`cmm_member_isfrozen()`	Determines whether a node has a `CMM_FROZEN_MEMBER` attribute.
`cmm_member_isoutofcluster()`	Determines whether a node has the `CMM_OUT_OF_CLUSTER` role. While a node has the `CMM_OUT_OF_CLUSTER` role, the qualification level and synchronization flag of the node are meaningless.
`cmm_member_isvicemaster()`	Determines whether a node has the `CMM_VICEMASTER` role.
`cmm_member_ismaster()`	Determines whether a node has the `CMM_MASTER` 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 43, and "Identifying the Role of a Node" on page 50 . See also the `cmm_member_iseligible`(3CMM) and `cmm_member_ismaster`(3CMM) man pages.

CHAPTER 6

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 57
- "Notifications During Changes in the Cluster State" on page 60

Introduction to Change Notifications

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

In a cluster, the master node is aware of all changes in the state of peer nodes. The cluster state information held by the `nhcmmd` 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 60.

For an example of how to retrieve notifications about changes in the cluster state, see Example 7–2.

To verify that the `nhcmmd` daemon is running on your peer nodes, see the *Netra High Availability Suite Foundation Services 2.1 6/03 Cluster Administration Guide*. For information about the `nhcmmd` daemon, see the `nhcmmd`(1M) 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 Example 6–1 details the related structure, called the `cmm_cmc_notification_t` structure.

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
nodeid	This field represents the node on which the change occurs if the notification is made for each node.
cmchange	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
change_notification	This parameter is a pointer to a structure describing the specific membership change and the affected cluster member's identity. This is either the *nodeid* of a specific node, or `0` when the change affects all peer nodes.
client_data	This parameter is a service or application-defined value given to the `cmm_cmc_register()` 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. The following table lists the notifications of the `cmm_cmchanges_t` structure:

TABLE 6–3 Change Notifications

Value	Description
`CMM_INVALID_CLUSTER`	A critical problem occurred. For example, there are two master nodes. One node must be rebooted as soon as possible. The *nodeid* field is not useful in this case.
`CMM_MASTER_DEMOTED`	The *nodeid* represents a previous master node that has been demoted. For more information, see "Administrative Attributes" on page 25.
`CMM_MASTER_ELECTED`	The *nodeid* 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.
`CMM_MEMBER_JOINED`	A peer node has joined the cluster. The *nodeid* is that of the new peer node.
`CMM_MEMBER_LEFT`	A peer node has the `CMM_OUT_OF_CLUSTER` role.

TABLE 6–3 Change Notifications *(Continued)*

Value	Description
`CMM_STALE_CLUSTER`	The master node sends a *membership frame* 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 *nodeid* 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 *nodeid* 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 *nodeid* is that of the newly elected vice-master node.
`CMM_VALID_STATE`	The state of the cluster is now valid and running correctly. The *nodeid* 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 `nhcmmd` 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 `nhcmmd` 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 23.

This section describes scenarios of cluster state change and these related notifications:

- "Cluster Initialization Notifications" on page 61.
- "Vice-Master Removal Notifications" on page 62.
- "Vice-Master Excluded Notification" on page 63.
- "Peer Node Removal Notification" on page 63.
- "Master Node Excluded Notifications" on page 64.
- "Node Other Than Master Excluded Notification" on page 64.
- "Switchover Notifications" on page 64.
- "Failover Notifications" on page 65.
- "Stale Cluster Notification" on page 66.
- "Amnesia" on page 67.
- "Split Brain" on page 67.

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 A Master is Elected at Cluster Startup

Transition (node A, node B)	Notifications Sent
(`out`, `out`) —> (`master`, `out`)	`CMM_MASTER_ELECTED`(A)

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 A New Node Joins the Cluster and Becomes Vice-Master

Transition (node A, node B)	Notifications Sent
(`master`, `out`) —> (`master`, `vice-master`)	`CMM_VICEMASTER_ELECTED` (B)

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 A New Node Joins the Cluster

Transition (node A, node B)	Notifications Sent
(`master`, `out`) —> (`master`, `in`)	`CMM_MEMBER_JOINED` (B)

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 A 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 65.

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 The Vice-Master Node Fails or the Vice-Master 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 The 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 77. 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 83. 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 A 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 76. The notification sent in this scenario is shown in Table 6–12.

TABLE 6–12 The 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 76. The notification sent is shown in Table 6–13.

TABLE 6–13 A 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 "Starting and Stopping Services, Nodes, and Clusters" in the *Netra High Availability Suite Foundation Services 2.1 6/03 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 A Switchover Triggered by the `cmm_mastership_release()` Function

Transition (node A, node B)	Notifications Sent
(`master`, `vice-master`) —> (`vice-master`, `master`)	`CMM_MASTER_ELECTED` (B) `CMM_VICEMASTER_ELECTED` (A)

For further information and an example that uses the `cmm_mastership_release()` function to trigger a switchover, see "Triggering A Switchover" on page 79.

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 65.
- "Failover Due to Master Disqualification" on page 66.

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 23.

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 A 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 57.

For an example of how to trigger a failover using the`cmm_membership_remove()` function, see Example 7–6.

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 A 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 57.

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 83.

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 The 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 Foundation Services 2.1 6/03 Troubleshooting Guide*.

CHAPTER 7

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 `nhcmmd` 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 `nhcmmd` daemon.
- Set the timeout value for subsequent calls to the `nhcmmd` 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 `nhcmmd` 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) 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:

EXAMPLE 7–1 Setting a Timeout Using the `cmm_connect()` Function

```
#include <cmm.h>
#include <stdio.h>
#include <stdlib.h>         /* for exit() */
#include <strings.h>         /* for strcpy */
#include "common.h"



/***************************************************************/
void main(void)
{
    cmm_error_t  res;
    timespec_t   time-out = { 1 /*seconds*/, 500000000 /* nanoseconds */};

    /* test connection and set time-out */
    if ((res = cmm_connect(timeout))!= CMM_OK) {
        printf("problem to connect to local CMM: %s -> abort\n",
            cmm_strerror(res));
        exit(1);
    }
    exit(0) ;
}
```

Reloading the Cluster Node Table

The `cmm_config_reload()` function can be called from the master node to make the `nhcmmd` 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 Foundation Services 2.1 6/03 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 `nhcmmd`(1M) 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

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 Foundation Services 2.1 6/03 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 section contains the following topics:

- "Registering to Receive Notifications" on page 72.
- "Filtering Notifications" on page 72.
- "Receiving and Dispatching Notifications" on page 72.
- "Retrieving Change Notifications" on page 73.

Registering to Receive Notifications

To receive notifications, applications or services can use the `cmm_cmc_register()` function to register a callback with the `nhcmmd` daemon. When a membership change occurs in the cluster, the `nhcmmd` 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 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 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 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:

EXAMPLE 7–2 Retrieving Change Notifications

```
#include <cmm.h>
#include <stdio.h>
#include <stdlib.h>        /* for exit() */
#include <strings.h>        /* for strcpy */
#include "common.h"

/*******************************************************************/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
void CMM_notify_cb(const cmm_cmc_notification_t *change_notification,
    void *client_data)
{
    cmm_member_t member ;

        switch (change_notification->cmchange)
    {
        case CMM_MASTER_ELECTED:
                printf("[USER CB] master elected = %d\n",
            change_notification->nodeid) ;
                if (cmm_master_getinfo(&member) == CMM_OK)
            print_member(&member) ;
                break ;
        case CMM_MASTER_DEMOTED:
                printf("[USER CB] master demoted = %d\n",
            change_notification->nodeid) ;
                break ;
        case CMM_VICEMASTER_ELECTED:
                printf("[USER CB] vicemaster elected = %d\n",
            change_notification->nodeid) ;
                break ;
        case CMM_VICEMASTER_DEMOTED:
                printf("[USER CB] vicemaster demoted = %d\n",
            change_notification->nodeid) ;
                break ;
        case CMM_MEMBER_JOINED:
                if (cmm_member_getinfo(change_notification->nodeid,
            &member) == CMM_OK)
            print_member(&member) ;
                break ;
        case CMM_MEMBER_LEFT:
                printf("[USER CB] member left cluster = %d\n",
            change_notification->nodeid) ;
```

EXAMPLE 7–2 Retrieving Change Notifications *(Continued)*

```
                break ;
        case CMM_INVALID_CLUSTER:
                printf("[USER CB] INVALID CLUSTER\n") ;
                break ;
        case CMM_STALE_CLUSTER:
                printf("[USER CB] STALE CLUSTER\n") ;
                break ;
        case CMM_VALID_CLUSTER:
                printf("[USER CB] VALID CLUSTER\n") ;
                break ;
        }
}


/*******************************************************************/
/*******************************************************************/
/*******************************************************************/
/*******************************************************************/
int main(int P_argc, char *P_argv[])
{
        uint32_t        L_node_count ;
        cmm_error_t     result ;
        struct pollfd   fifo_poll ;
    boolean_t       go_on_poll = B_TRUE ;

    /* FILTERING PART */
    cmm_cmchanges_t L_notify[2] = { CMM_MASTER_ELECTED,
                    CMM_INVALID_CLUSTER } ;

    cmm_cmc_filter(CMM_CMC_NOTIFY_SET, L_notify, 2) ;

    result = cmm_cmc_register(CMM_notify_cb, (void *) NULL) ;
    if (result != CMM_OK)
    {
        fprintf(stderr,
            " register error (%s) => EXIT\n",
            cmm_strerror(result)) ;
        exit(1) ;
    }

    result = cmm_notify_getfd(&fifo_poll.fd) ;
    if (result != CMM_OK) {
        fprintf(stderr,
            " getfd error (%s) => EXIT\n",
            cmm_strerror(result)) ;
        exit(1) ;
    }

    printf("file descriptor to poll = %d\n", fifo_poll.fd) ;
    fifo_poll.events
        = (POLLIN | POLLRDNORM | POLLRDBAND | POLLPRI);
    fifo_poll.revents = 0 ;
    while (1) {
        /* this loop ends when exit asked or poll succeeded */
        go_on_poll = B_TRUE ;
        while (go_on_poll == B_TRUE) {
            result = cmm_notify_getfd(&fifo_poll.fd) ;
            switch (poll(&fifo_poll, 1, 2500) )
            {
            case -1:
                puts("error on poll") ;
                break ;
            case 0:
                /* time-out */
                break ;
            default:
                go_on_poll = B_FALSE ;
            }

            if ((fifo_poll.revents & POLLHUP) ||
                (fifo_poll.fd == -1))
            {
                printf("reconnection required\n") ;
                result = cmm_cmc_unregister() ;
                result = cmm_cmc_register(CMM_notify_cb,
                    (void *) NULL) ;
                result = cmm_notify_getfd(&fifo_poll.fd) ;
                poll(NULL, 0 , 500) ;
            }
        }
        cmm_notify_dispatch() ;
    }
}
```

Responding to Cluster Notifications by Modifying the Cluster

You can use the CMM API to respond to notifications received from the `nhcmmd` 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 section contains these topics:

- "Removing or Excluding a Node" on page 76
- "Setting the Qualification of a Node" on page 77
- "Triggering A Switchover" on page 79

- "Triggering A Failover" on page 81

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 23.

If you want to exclude a node completely from the cluster, first use the `cmm_membership_remove()` function to remove the role 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 81.

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 79. For an explanation of the notifications sent when you trigger a switchover, see "Switchover Notifications" on page 64.

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 81.

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 65.

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 Foundation Services 2.1 6/03 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 a Diskless Node

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.

EXAMPLE 7–3 Disqualifying a Node

```
if (cmm_member_setqualif (12, CMM_DISQUALIFIED_MEMBER) != CMM_OK)
                    /* handle the error  */;
```

The following example demonstrates how to use the `cmm_node_getid()` and `cmm_member_setqualif()` functions.

EXAMPLE 7–4 Requalifying a Node

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

```
#include <sys/types.h>       /* for boolean_t */
#include <stdlib.h>          /* for exit(), atoi() */
#include "common.h"

/******************************************************************/
int get_id(cmm_nodeid_t *P_NodeId)
{
    int          success;
    char         str_node[10];
    boolean_t    go_on = B_TRUE;

    while (go_on == B_TRUE) {
        printf("Your node [0 for abort]: ");
        success = (scanf("%9s", str_node) != 0);
        if (success) {
            *P_NodeId = atoi(str_node);
            if (*P_NodeId >= 0)
                go_on = FALSE;
        }
    }
    return (*P_NodeId==0)?FALSE:TRUE;
}
/*******************************************************************/
/*******************************************************************/
int ask_for_qualif(cmm_qualif_t *P_NewQualif)
{
    char         L_Choice[20] ;

    puts("Your qualif:  o Q  qualif") ;
    puts("              o D  disqualif")
    printf("your choice: ") ;
    scanf("%6s", L_Choice) ;
    if (strcasecmp(L_Choice,"Q") == 0)
        *P_NewQualif = CMM_QUALIFIED_MEMBER ;
    else if (strcasecmp(L_Choice,"D") == 0)
        *P_NewQualif = CMM_DISQUALIFIED_MEMBER ;
    else
    {
        printf("bad qualif: [%s]",L_Choice ) ;
        return FALSE ;
    }
    return TRUE ;
}

/*******************************************************************/
void main(void)
{
    cmm_error_t  res;
    cmm_nodeid_t onenode;
    cmm_qualif_t newqualif;

    if (!get_id(&onenode))    {
        printf("abort\n");
        exit(0);
    }

    if (!ask_for_qualif(&newqualif)) {
        printf("abort\n");
        exit(0);
    }

    if ((res = cmm_member_setqualif(onenode, newqualif)) != CMM_OK) {
        fprintf(stderr,
            "ERROR: Failed to set qualif (Error: %s)\n",
            cmm_strerror(res));
    }
}
```

You can trigger a failover by calling the `cmm_member_setqualif()` function from the master node. For more information, see "Triggering a Failover by Using the `cmm_member_setqualif()` Function" on page 83.

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 `nhcmmd` daemon issues a notification of the switchover. For information about notifications during a switchover, see "Switchover Notifications" on page 64. For more information about the `nhcmmd` daemon and notifications, see "Introduction to Change Notifications" on page 57.

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.

Example 7–5 demonstrates how to use the `cmm_mastership_release()` function to trigger a switchover:

EXAMPLE 7–5 Triggering a Switchover by Using the `cmm_mastership_release()` Function

```
#include <cmm.h>
#include <stdio.h>
#include "common.h"

/**************************************************************/
void main(void)
{
    cmm_error_t  res;
    cmm_member_t master_info;
    cmm_nodeid_t currnode;

    /* get the current node id */
    if ((res = cmm_node_getid(&currnode))==CMM_OK)
        printf("Current node id is: %d\n", currnode);
    else {
        printf("Error getting info on local node: %s\n",
            cmm_strerror(res));
        exit(1) ;
    }

    /* get the master information */
    if ((res = cmm_master_getinfo(&master_info)) != CMM_OK) {
        printf("Error getting info on master: %s\n",
            cmm_strerror(res));
        exit(1);
    }

    print_member(&master_info);
    /* get the current node id */

    if (master_info.nodeid != currnode)
    {
        printf("you must be on master to execute this command\n") ;
        exit(1);
    }

    /* note that cmm_mastership_release internally checks that
     * the command is executed on master node */
    res = cmm_mastership_release();
    if (res != CMM_OK) {
        printf("Error to release masterhsip: %s\n",
            cmm_strerror(res));
        exit(1) ;
    }
    printf("Switch-over completed successfully\n") ;
    exit(0) ;
```

EXAMPLE 7–5 Triggering a Switchover by Using the `cmm_mastership_release()` Function *(Continued)*

```
}
```

Note – Do not trigger a switchover with the `cmm_mastership_release()` function if you are running a separate Netra HA Suite Framework product on your cluster.

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.

This section contains information about these topics:

- "Triggering a Failover by Using the `cmm_membership_remove()` Function" on page 81.
- "Triggering a Failover by Using the `cmm_member_setqualif()` Function" on page 83.

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 65.

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 89.

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) man page.

An example that demonstrates how to trigger a failover by using the `cmm_membership_remove()` function is shown in Example 7–6.

EXAMPLE 7–6 Triggering a Failover Using the `cmm_membership_remove()` Function

```
#include <cmm.h>
#include <stdio.h>
#include "common.h"

/**************************************************************/
void main(void)
{
    cmm_error_t  res;
    cmm_member_t master_info;
    cmm_nodeid_t currnode;

    /* get the current node id */
    if ((res = cmm_node_getid(&currnode))==CMM_OK)
        printf("Current node id is: %d\n", currnode);
    else {
        printf("Error getting info on local node: %s\n",
            cmm_strerror(res));
        exit(1) ;
    }

    /* get the master information */
    if ((res = cmm_master_getinfo(&master_info)) != CMM_OK) {
        printf("Error getting info on master: %s\n",
            cmm_strerror(res));
        exit(1);
    }

    print_member(&master_info);
    /* get the current node id */

    if (master_info.nodeid != currnode)
    {
        printf("you must be on master to trigger a failover
        using the cmm_membership_remove command\n") ;
        exit(1);
    }


    res = cmm_membership_remove();
    if (res != CMM_OK) {
        printf("Error in triggering a failover: %s\n",
            cmm_strerror(res));
        exit(1) ;
    }
```

EXAMPLE 7–6 Triggering a Failover Using the `cmm_membership_remove()` Function *(Continued)*

```
    printf("Failover successful\n") ;
    exit(0) ;
}
```

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. An example that demonstrates how to trigger a failover using the `cmm_member_setqualif()` function is shown in Example 7–7.

EXAMPLE 7–7 Triggering a Failover Using the `cmm_member_setqualif()` Function

```
#include <stdio.h>
#include <unistd.h>
#include <cmm.h>
#include "common.h"

/**********************************************************/
int main(int argc , char ** argv) {
    cmm_error_t  cmm_diag;
    cmm_nodeid_t curr_node_id;
    cmm_member_t master;
    cmm_member_t vicemaster;

    cmm_diag = cmm_node_getid(&curr_node_id);
    if (cmm_diag != CMM_OK) {
        fprintf(stderr,
            "An error occurred during cmm_node_getid() call \n");
        exit(1);
    }

    printf("My id is %d\n",curr_node_id);


    /* get master/vice master id */
    cmm_diag = cmm_master_getinfo(&master);
    if (cmm_diag != CMM_OK) {
        fprintf(stderr,
            "An error occurred during cmm_master_getinfo call, CR=%d\n",
            cmm_diag);
        exit(1);
    }

    cmm_diag = cmm_vicemaster_getinfo(&vicemaster);
    if (cmm_diag != CMM_OK) {
        fprintf(stderr,
            "An error occurred during cmm_vicemaster_getinfo call, CR=%d\n",
```

EXAMPLE 7–7 Triggering a Failover Using the `cmm_member_setqualif()` Function *(Continued)*

```
            cmm_diag);
        exit(1);
    }

    printf("master node id is %d\n",master.nodeid);
    printf("vice-master node id is %d\n",vicemaster.nodeid);

    /* the role could be verified by cmm_member_ismaster() */
    if (curr_node_id == master.nodeid) {
        printf("We are the master of the cluster, launch Failover\n");
    } else {
        printf("We are not master, operation will be cancelled\n");
        exit(1);
    }

    /* launch the failover (disqualify the master) */
    cmm_diag = cmm_member_setqualif(curr_node_id,CMM_DISQUALIFIED_MEMBER);
    if (cmm_diag != CMM_OK) {
        fprintf(stderr,
            "An error occurred during cmm_member_setqualif, CR=%d\n",
            cmm_diag);
        exit(1);
    }

    /*the failover is running*/
    /* ... */


    /* on the new master we should run this command to
     * qualify the ex-master to make it the vice-master of the cluster
     *
     *  cmm_diag = cmm_member_setqualif(curr_node_id,CMM_QUALIFIED_MEMBER);
     *  if (cmm_diag != CMM_OK) {
     *      fprintf(stderr,
     *             "An error occurred during cmm_member_setqualif, CR=%d\n",
     *             cmm_diag);
     *      exit(1);
     *  }
     */
    exit(0);
}
```

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.

CHAPTER 8

Debugging Applications in the Foundation Services

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

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 Foundation Services 2.1 6/03 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 Forte Developer 6 Software Suite.

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.

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 Foundation Services 2.1 6/03 NMA Programming Guide*.

For information about using and configuring system log files, see the *Netra High Availability Suite Foundation Services 2.1 6/03 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 Foundation Services 2.1 6/03 Cluster Administration Guide*. For a list of monitored daemons, see the `nhpmd`(1M) 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 `nhcmmd` 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 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 Foundation Services 2.1 6/03 Cluster Administration Guide* and the `syslog.conf`(4) Solaris 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.

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

Return Value	Result	Possible Responses
`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 `cmm_cmc_register()` and the `cmm_notify_getfd()` functions to fetch a new connection.
`CMM_EBUSY`	■ For all functions: The CMM API server is temporarily out of resources to respond to the requested operation. ■ For `cmm_cmc_unregister()`: An attempt to unregister a callback, that is, a call to the `cmm_cmc_unregister()` function, failed because the caller's callback function is active. See the `cmm_cmc_unregister`(3CMM) 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 `cmm_cmc_register()` 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.
`CMM_EINVAL`	A function parameter has an invalid value.	Ensure that the type of each parameter matches the type in the function prototype. For example the *nodeid* is not a master-eligible node. Cast variables to the expected type if necessary and verify that the area of memory that stores the parameter is valid.

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

Return Value	Result	Possible Responses
`CMM_ENOCLUSTER`	One of the following has occurred: ■ 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 `cluster_nodes_table`(4) 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.	Any combination of the following: ■ 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 `cmm_cmc_unregister()` function, no callback has been registered. Not critical.	Any combination of the following: ■ 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.
`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 `cmm_mastership_release`(3CMM), `cmm_member_setqualif`(3CMM), and `cmm_member_seizequalif`(3CMM) man pages.	Execute the function only on the master node.

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

Return Value	Result	Possible Responses
`CMM_ERANGE`	The number of cells in the table is smaller than the number of nodes in the cluster. Returned by the `cmm_member_getall()` function. See the `cmm_member_getall`(3CMM) man page.	Add an entry in the table for each potential peer node.
`CMM_ESRCH`	■ Using the `cmm_member_getinfo()` 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 `CMM_OUT_OF_CLUSTER` role. ■ Using the `cmm_potential_getinfo()` function to obtain information about a node that is not in the local cluster node table. ■ Using the `cmm_vicemaster_getinfo()` function while the cluster has no vice-master.	Any combination of the following: ■ Examine why the master-eligible node is down or isolated. ■ Add an entry for this node to the cluster node table. See the `cluster_nodes_table`(4) man page. ■ Change the node's role to master or vice-master.
`CMM_ETIMEDOUT`	No response even when an operation is retried, until the delay has expired. The function call was timed out.	Any combination of the following: ■ Retry the function call. ■ Reduce the load on the system.

APPENDIX **A**

Source Code Examples

The Foundation Services product provides a set of source code examples in the `SUNWnhcmd` developer package. The examples are described in detail in the following sections:

- "Example Makefile" on page 93
- "The `cmm_master_getinfo()` Function" on page 95
- "The `cmm_member_getcount()` Function" on page 96
- "The `cmm_member_getall()` Function" on page 98
- "The `cmm_member_getinfo()` Function" on page 99
- "The `cmm_member_setqualif()` Function" on page 100
- "The `cmm_node_getid()` Function" on page 102
- "The `cmm_vicemaster_getinfo()` Function" on page 103
- "CMM API Extended Code Example" on page 104

CMM API Code Examples

The examples in this section use the functions of the CMM API. These examples are located in subdirectories of the `/opt/SUNWcgha/examples/` directory. You can compile these examples on your development host and test the programs on your cluster.

Before running these code examples, create the following Makefile:

Example Makefile

The Makefile given in Example A–1 is provided with the `SUNWcgha` package for use with the examples provided within this appendix.

EXAMPLE A–1 Sample Makefile

```
#
# Copyright (c) 2002 by Sun Microsystems, Inc.
# All rights reserved.
#
#
# ident  "@(#)Makefile.ex 1.4     02/06/05 SMI"
#
#


############################################################
# Path to the directory where SUNWnhcmd package is installed
############################################################
NHCMMD_PKG_INSTALL_DIR=/opt/SUNWcgha

LDFLAGS= -L"$(NHCMMD_PKG_INSTALL_DIR)/lib" \
     -lcgha_cmm                      \
     -R"$(NHCMMD_PKG_INSTALL_DIR)/lib" \
     -lpthread

CFLAGS= -I"/usr/local/include"       \
    -I$(NHCMMD_PKG_INSTALL_DIR)/include/
CC=cc

COMPILE=$(CC) $(CFLAGS) $(LDFLAGS)

BINS=   smpl_cmm_node_getid            \
    smpl_cmm_member_getinfo        \
    smpl_cmm_master_getinfo        \
    smpl_cmm_vicemaster_getinfo    \
    smpl_cmm_member_getcount_all   \
    smpl_cmm_member_getcount_all_2 \
    smpl_cmm_notification          \
    smpl_cmm_member_setqualif


all:$(BINS)


smpl_cmm_node_getid:smpl_cmm_node_getid.c
    @echo "CC smpl_cmm_node_getid.c"
    @$(COMPILE) smpl_cmm_node_getid.c -o smpl_cmm_node_getid

smpl_cmm_member_getinfo:smpl_cmm_member_getinfo.c
    @echo "CC smpl_cmm_member_getinfo.c"
    @$(COMPILE) smpl_cmm_member_getinfo.c -o smpl_cmm_member_getinfo

smpl_cmm_master_getinfo:smpl_cmm_master_getinfo.c
    @echo "CC smpl_cmm_master_getinfo.c"
    @$(COMPILE) smpl_cmm_master_getinfo.c -o smpl_cmm_master_getinfo

smpl_cmm_vicemaster_getinfo:smpl_cmm_vicemaster_getinfo.c
    @echo "CC smpl_cmm_vicemaster_getinfo.c"
```

EXAMPLE A–1 Sample Makefile *(Continued)*

```
    @$(COMPILE) smpl_cmm_vicemaster_getinfo.c -o smpl_cmm_vicemaster_getinfo

smpl_cmm_member_getcount_all:smpl_cmm_member_getcount_all.c
    @echo "CC smpl_cmm_member_getcount_all.c"
    @$(COMPILE) smpl_cmm_member_getcount_all.c
    -o smpl_cmm_member_getcount_all

smpl_cmm_member_getcount_all_2:smpl_cmm_member_getcount_all_2.c
    @echo "CC smpl_cmm_member_getcount_all_2.c"
    @$(COMPILE) smpl_cmm_member_getcount_all_2.c
    -o smpl_cmm_member_getcount_all_2

smpl_cmm_notification:smpl_cmm_notification.c
    @echo "CC smpl_cmm_notification.c"
    @$(COMPILE) smpl_cmm_notification.c -o smpl_cmm_notification

smpl_cmm_member_setqualif:smpl_cmm_member_setqualif.c
    @echo "CC smpl_cmm_member_setqualif.c"
    @$(COMPILE) smpl_cmm_member_setqualif.c -o smpl_cmm_member_setqualif


clean:
    rm -f core *~

cleanall:clean
    rm -f $(BINS)
```

The `cmm_master_getinfo()` Function

The code provided by Example A–2 gets information about master node by using the `cmm_master_getinfo()` function.

EXAMPLE A–2 The `cmm_master_getinfo.c` Program

```
/*********************************************************
* Copyright (c) 2002 by Sun Microsystems, Inc.
* All rights reserved.
*
*
* ident  "@(#)smpl_cmm_master_getinfo.c 1.2     02/06/05 SMI"
*
****************************************************************************/

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

int main(int argc , char **argv) {
  cmm_error_t  cmm_diag;
  cmm_member_t nodeInfo;
  cmm_diag = cmm_master_getinfo(&nodeInfo);

  if (cmm_diag != CMM_OK) {
    fprintf(stderr,"An error occured during
    cmm_member_getinfo call, CR=%d\n",cmm_diag);
    fprintf(stderr,"Details: %s\n",cmm_strerror(cmm_diag));
    exit(1);
  }

  printf("Infornation on Master:\n");
  printf("\tName:               %s\n",nodeInfo.name);
  printf("\tAdress:             %s\n",nodeInfo.addr);
  printf("\tDomain Id:          %d\n",nodeInfo.domainid);
  printf("\tIncarnation number: %d\n",nodeInfo.incarnation_number);
  printf("\tSoftwareLoad id:    %s\n",nodeInfo.software_load_id);


  exit(0);
}
```

The `cmm_member_getcount()` Function

The code provided by Example A–3 gets information about all nodes in the cluster by using the `cmm_member_getcount()` function.

EXAMPLE A–3 The `smpl_cmm_member_getcount_all.c` Program

```
/*********************************************************
* Copyright (c) 2002 by Sun Microsystems, Inc.
* All rights reserved.
*
*
* ident  "@(#)smpl_cmm_member_getcount_all.c 1.3     02/09/25 SMI"
*
**************************************************************************/

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

int main(int argc , char ** argv) {
  cmm_error_t   L_Status;
  uint32_t      L_NodesNumber;
  cmm_member_t *L_Table = NULL;
  uint32_t      L_TableSize;
  int           i;

  /*Getting node count with cmm_member_getcount() */
  L_Status = cmm_member_getcount(&L_NodesNumber);
  if (L_Status != CMM_OK) {
    fprintf(stderr,"cmm_member_getcount() error: %s\n",
    cmm_strerror(L_Status));
    exit(1);
  }

  printf("Number of nodes in cluster: %d\n",L_NodesNumber);

  /*Getting information on all nodes in cluster*/
  L_TableSize = L_NodesNumber + 3; /* "+3" to be safer */

  L_Table = (cmm_member_t *) malloc(L_TableSize * sizeof(cmm_member_t));
  if (L_Table == NULL) {
    fprintf(stderr,"Memory allocation error\n");
    exit(1);
  }


  L_Status = cmm_member_getall(L_TableSize,L_Table,&L_NodesNumber);

  if (L_Status != CMM_OK) {
    fprintf(stderr,"cmm_member_getall() error %s\n",
    cmm_strerror(L_Status));
    free(L_Table);
    exit(1);
  }

  printf("Information on cluster:\n");
  for (i = 0 ; i < L_NodesNumber ; i++) {
    printf("Infornation on node:  %d\n",L_Table[i].nodeid);
    printf("\tName:               %s\n",L_Table[i].name);
    printf("\tAdress:             %s\n",L_Table[i].addr);
    printf("\tDomain Id:          %d\n",L_Table[i].domainid);
    printf("\tIncarnation number: %d\n",L_Table[i].incarnation_number);
    printf("\tSoftwareLoad id:    %s\n",L_Table[i].software_load_id);
    printf("\tRole:               ");
    if      (cmm_member_ismaster(&L_Table[i]))     printf("MASTER\n");
    else if (cmm_member_isvicemaster(&L_Table[i]))
            printf("VICE-MASTER\n");
    else    printf("IN CLUSTER\n");

    printf("\tQualification:      ");
    if      (cmm_member_isqualified(&L_Table[i]))    printf("QUALIFIED\n");
    else if (cmm_member_isdisqualified(&L_Table[i]))
            printf("DISQUALIFIED\n");

  }

  free(L_Table);

  exit(0);
}
```

The `cmm_member_getall()` Function

The code provided by Example A–4 gets information about all nodes in the cluster by using the `cmm_member_get_all()` function.

EXAMPLE A–4 The `smpl_cmm_member_get_all.c` Program

```
/****************************************************************
* Copyright (c) 2002 by Sun Microsystems, Inc.
* All rights reserved.
*
*
* ident  "@(#)smpl_cmm_member_getcount_all_2.c 1.2     02/06/05 SMI"
*
****************************************************************/


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



#define MAX_NODE_IN_CLUSTER 256

int main(int argc , char ** argv) {
  cmm_error_t   cmm_diag;
  uint32_t      nbNodeInCluster;
  cmm_member_t *NodeTable = NULL;
  int           i;



  /*allocate space for the maximum number of nodes*/
NodeTable =
  (cmm_member_t *)malloc(MAX_NODE_IN_CLUSTER*sizeof(cmm_member_t));
  if (NodeTable == NULL) {
    fprintf(stderr,"Memory allocation error\n");
    exit(1);
  }

  /*Getting information on all nodes in cluster*/

cmm_diag =
  cmm_member_getall(MAX_NODE_IN_CLUSTER,NodeTable,&nbNodeInCluster);

  if (cmm_diag != CMM_OK) {
    fprintf(stderr,"An error occured during
    cmm_member_getall() call CR=%d\n",cmm_diag);
    fprintf(stderr,"Details: %s\n",cmm_strerror(cmm_diag));
    free(NodeTable);
    exit(1);
  }

  /*free useless space*/
NodeTable =
(cmm_member_t *)realloc(NodeTable,nbNodeInCluster*sizeof(cmm_member_t));
  if (NodeTable == NULL) {
    fprintf(stderr,"Memory allocation error\n");
    exit(1);
  }


  printf("Information on cluster:\n");
  for (i = 0 ; i < nbNodeInCluster ; i++) {
    printf("Infornation on node:  %d\n",NodeTable[i].nodeid);
    printf("\tName:               %s\n",NodeTable[i].name);
    printf("\tAdress:             %s\n",NodeTable[i].addr);
    printf("\tDomain Id:          %d\n",NodeTable[i].domainid);
    printf("\tIncarnation number: %d\n",NodeTable[i].incarnation_number);
    printf("\tSoftwareLoad id:    %s\n",NodeTable[i].software_load_id);
    printf("\tRole:               ");
    if      (cmm_member_ismaster(&NodeTable[i]))     printf("MASTER\n");
    else if (cmm_member_isvicemaster(&NodeTable[i]))
    printf("VICE-MASTER\n");
    else printf("IN CLUSTER\n");

    printf("\tQualification:      ");
    if      (cmm_member_isqualified(&NodeTable[i]))
    printf("QUALIFIED\n");
    else if (cmm_member_isdisqualified(&NodeTable[i]))
    printf("DISQUALIFIED\n");

  }

  free(NodeTable);

  exit(0);
}
```

The `cmm_member_getinfo()` Function

The code provided by Example A–5 gets information about a named node, in this case `node 10`, by using the `cmm_member__getinfo()` function.

EXAMPLE A–5 The `smpl_cmm_member_getinfo.c` Program

```
/*********************************************************
* Copyright (c) 2002 by Sun Microsystems, Inc.
* All rights reserved.
*
*
* ident  "@(#)smpl_cmm_member_getinfo.c 1.2     02/06/05 SMI"
*
*************************************************************************/
```

EXAMPLE A–5 The `smpl_cmm_member_getinfo.c` Program *(Continued)*

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

int main(int argc , char **argv) {
  cmm_error_t  cmm_diag;
  cmm_nodeid_t node;
  cmm_member_t nodeInfo;

  node = 10;

  /*getting info for node 10*/

  cmm_diag = cmm_member_getinfo(node,&nodeInfo);

  if (cmm_diag != CMM_OK) {
  fprintf(stderr,"An error occured during
  cmm_member_getinfo call, CR=%d: %s\n",cmm_diag,cmm_strerror(cmm_diag));
    exit(1);
  }

  printf("Infornation on node:  %d\n",node);
  printf("\tName:                 %s\n",nodeInfo.name);
  printf("\tAdress:               %s\n",nodeInfo.addr);
  printf("\tDomain Id:            %d\n",nodeInfo.domainid);
  printf("\tIncarnation number: %d\n",nodeInfo.incarnation_number);
  printf("\tSoftwareLoad id:    %s\n",nodeInfo.software_load_id);
  printf("\tRole:                 ");
  if      (cmm_member_ismaster(&nodeInfo))     printf("MASTER\n");
  else if (cmm_member_isvicemaster(&nodeInfo)) printf("VICE-MASTER\n");
  else                                         printf("IN CLUSTER\n");

  printf("\tQualification:        ");
  if      (cmm_member_isqualified(&nodeInfo))    printf("QUALIFIED\n");
  else if (cmm_member_isdisqualified(&nodeInfo)) printf("DISQUALIFIED\n");


  exit(0);
}
```

The `cmm_member_setqualif()` Function

The code provided by Example A–6 performs a switchover by disqualifying the master node by using the `cmm_member_setqualif()` function.

EXAMPLE A–6 The `smpl_cmm_member_setqualif.c` Program

```
/**********************************************************
* Copyright (c) 2002 by Sun Microsystems, Inc.
* All rights reserved.
```

```
*
*
* ident  "@(#)smpl_cmm_member_setqualif.c 1.2     01/12/14 SMI"
**************************************************************************/


#include <stdio.h>
#include <unistd.h>

#include <cmm.h>

/*
 * get master ID
 * test if we are on master
 * dequalify ourself
 *
 */

int main(int argc , char ** argv) {
  cmm_error_t  cmm_diag;
  cmm_nodeid_t CurrentNodeid;
  cmm_member_t MasterNodeInfo;
  cmm_member_t ViceMasterNodeInfo;

  cmm_diag = cmm_node_getid(&CurrentNodeid);
  if (cmm_diag != CMM_OK) {
    fprintf(stderr,"An error occured during cmm_node_getid() call \n");
    fprintf(stderr,"Details: %s\n",cmm_strerror(cmm_diag));
    exit(1);
  }

  printf("My id is %d\n",CurrentNodeid);


  /*get master/vice master id*/
  cmm_diag = cmm_master_getinfo(&MasterNodeInfo);
  if (cmm_diag != CMM_OK) {
    fprintf(stderr,"An error occured during
    cmm_master_getinfocall, CR=%d\n",cmm_diag);
    fprintf(stderr,"Details: %s\n",cmm_strerror(cmm_diag));
    exit(1);
  }

  cmm_diag = cmm_vicemaster_getinfo(&ViceMasterNodeInfo);
  if (cmm_diag != CMM_OK) {
    fprintf(stderr,"An error occured during
    cmm_master_getinfocall, CR=%d\n",cmm_diag);
    fprintf(stderr,"Details: %s\n",cmm_strerror(cmm_diag));
    exit(1);
  }

  printf("master node id is %d\n",MasterNodeInfo.nodeid);
  printf("vice-master node id is %d\n",ViceMasterNodeInfo.nodeid);
```

EXAMPLE A–6 The `smpl_cmm_member_setqualif.c` Program *(Continued)*

```
  /*the role could be verified by cmm_member_ismaster()*/
  if (CurrentNodeid == MasterNodeInfo.nodeid) {
    printf("We are the master of the cluster, launch the SO\n");
  } else {
    printf("We are not master, operation will be cancelled\n");
    exit(1);
  }

  /*disqualify the master*/
  cmm_diag =
   cmm_member_setqualif(CurrentNodeid,CMM_DISQUALIFIED_MEMBER);
  if (cmm_diag != CMM_OK) {
    fprintf(stderr,"An error occured during
    cmm_member_setqualif, CR=%d\n",cmm_diag);
    fprintf(stderr,"Details: %s\n",cmm_strerror(cmm_diag));
    exit(1);
  }

  /*the switch over is running*/
  /* ... */

  /* the master (ex master) is now disqualified                         */
  /* We need to be on the new master to be allowed to qualify a node    */
  /* on the new master we should run this command to                   */
  /* qualify the ex-master to make it the vice-master of the cluster    */
  /*                                                                    */
  /* cmm_diag =
  /*  cmm_member_setqualif(CurrentNodeid,CMM_QUALIFIED_MEMBER); */
  /*  if (cmm_diag != CMM_OK) {                                         */
  /*      fprintf(stderr,"An error occured during
  cmm_member_setqualif, CR=%d\n",cmm_diag);                             */
  /*      exit(1);                                                      */
  /*  }                                                                 */





  exit(0);

}
```

The `cmm_node_getid()` Function

The code provided by Example A–7 displays information about the current node by using the `cmm_node_getid()` function.

EXAMPLE A–7 The `smpl_cmm_node_getid.c` Program

```
/*********************************************************
* Copyright (c) 2002 by Sun Microsystems, Inc.
```

EXAMPLE A–7 The `smpl_cmm_node_getid.c` Program *(Continued)*

```
* All rights reserved.
*
*
* ident  "@(#)smpl_cmm_node_getid.c 1.2     02/06/05 SMI"
*
*********************************************************/

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

int main(int argc , char **argv) {
  cmm_nodeid_t nodeid;
  cmm_error_t cmm_diag;

  cmm_diag = cmm_node_getid(&nodeid);
  switch (cmm_diag) {
  case CMM_OK:
    {
      printf("The node id of this node is: %d\n",nodeid);
      exit(0);
    }
  case CMM_ETIMEDOUT:
    {
      fprintf(stderr,"Cmm didn't give us reponse within delay\n");
      exit(1);
    }
  case CMM_ECONN:
    {
      fprintf(stderr,"Cannot reach the CMM\n");
      exit(1);
    }
  default:
    fprintf(stderr,"call returned an error. CR=%d\n",cmm_diag);
    exit(1);
  }

  /*NOTREACHED*/
  exit(0);
}
```

The `cmm_vicemaster_getinfo()` Function

The code provided by Example A–8 displays information about the vice-master node by using the `cmm_vicemaster_getinfo()` function.

EXAMPLE A–8 The `smpl_cmm_vicemaster_getinfo.c` Program

```
/*********************************************************
* Copyright (c) 2002 by Sun Microsystems, Inc.
* All rights reserved.
```

EXAMPLE A–8 The `smpl_cmm_vicemaster_getinfo.c` Program *(Continued)*

```
*
*
* ident  "@(#)smpl_cmm_vicemaster_getinfo.c 1.2     02/06/05 SMI"
*
************************************************************/


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

int main(int argc , char **argv) {
  cmm_error_t  cmm_diag;
  cmm_member_t nodeInfo;


  cmm_diag = cmm_vicemaster_getinfo(&nodeInfo);

  if (cmm_diag != CMM_OK) {
    fprintf(stderr,"An error occured during
    cmm_member_getinfo call, CR=%d: %s\n",cmm_diag,cmm_strerror(cmm_diag));
    exit(1);
  }

  printf("Infornation on vice-master:\n");
  printf("\tName:              %s\n",nodeInfo.name);
  printf("\tAdress:            %s\n",nodeInfo.addr);
  printf("\tDomain Id:         %d\n",nodeInfo.domainid);
  printf("\tIncarnation number: %d\n",nodeInfo.incarnation_number);
  printf("\tSoftwareLoad id:    %s\n",nodeInfo.software_load_id);


  exit(0);
}
```

CMM API Extended Code Example

In addition to the examples and Makefile outlined in "CMM API Code Examples" on page 93, the developer package subdirectory `opt/SUNWcgha/examples/` provides an extended code example, which uses many of the function calls of the CMM API. This program reacts to the `CMM_MASTER_ELECTED` and `CMM_MASTER_DEMOTED` notifications, as demonstrated in the Example A–9.

EXAMPLE A–9 The `smpl_cmm_notification.c` Program

```
/*****************************************************
* Copyright (c) 2002 by Sun Microsystems, Inc.
* All rights reserved.
```

EXAMPLE A–9 The `smpl_cmm_notification.c` Program *(Continued)*

```
*
*
* ident  "@(#)smpl_cmm_notification.c 1.2     02/06/05 SMI"
*
*******************************************************/


#include <stdio.h>
#include <unistd.h>

#include <pthread.h>
#include <cmm.h>



/*
 * create a pthread to receive events from CMM
 * the thread will be activated on vice-master election
 */


typedef struct thrArg {
  int FilterChange;
  cmm_cmcfilter_t cmcFilter;
  cmm_cmchanges_t cmcChangeList[5];
  uint32_t        cmcChangeCount;
} thrArg_t;



void notification_thread_cb
(const cmm_cmc_notification_t * change_notification,void *client_data);

void CleanHandler(void *arg) {
  fprintf(stderr,"%d: I was cancelled\n",pthread_self());
  pthread_exit((void*)arg);
}


void *notification_thread(void*args) {
  struct pollfd pfd;
  int diag;
  cmm_error_t cmm_diag;

  thrArg_t *TArgs = (thrArg_t *)args;


  diag = pthread_setcancelstate(PTHREAD_CANCEL_ENABLE,NULL);
  if (diag != 0) {
    fprintf(stderr,"%d:
     call of pthread_setcancelstate failed\n",pthread_self());
    pthread_exit((void*)NULL);
  }
```

```
diag = pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED,NULL);
if (diag != 0) {
fprintf(stderr,"%d:
call of pthread_setcanceltype failed\n",pthread_self());
  pthread_exit((void*)NULL);
}


pthread_cleanup_push(CleanHandler,NULL);

/*register the callback, called after event are received*/

diag = cmm_cmc_register(notification_thread_cb,NULL);
if (diag != CMM_OK) {
  fprintf(stderr,"%d: error in
  cmm_cmc_register CR:%d:
  %s\n",pthread_self(),diag,cmm_strerror(cmm_diag));
  pthread_exit((void*)NULL);
}

diag = cmm_notify_getfd(&(pfd.fd));
if (diag != CMM_OK) {
  fprintf(stderr,"%d: error in
  cmm_notify_getfd CR:%d:
  %s\n",pthread_self(),diag,cmm_strerror(cmm_diag));
  cmm_cmc_unregister();
  pthread_exit((void*)NULL);
}

pfd.events = POLLIN;

while(1) {
  diag = poll(&pfd,1,1000);
  if (diag < 0) {
    fprintf(stderr,"%d:
    error during polling CR:%d\n",pthread_self(),diag);
    cmm_cmc_unregister();
    pthread_exit((void*)NULL);
  }

  /*we receive something*/
  if (pfd.revents == POLLIN) {
    cmm_diag = cmm_notify_dispatch();
    if (cmm_diag != CMM_OK) {
  fprintf(stderr,"%d: error during
  call of dispatch() CR:%d\n",pthread_self(),cmm_diag);
  fprintf(stderr,"Details: %s\n",cmm_strerror(cmm_diag));
  cmm_cmc_unregister();
  pthread_exit((void*)NULL);
    }
  }
  if (TArgs->FilterChange) {
    /*we have to change the filter*/
```

EXAMPLE A–9 The `smpl_cmm_notification.c` Program *(Continued)*

```
      printf("%d: we have to update the filter\n",pthread_self());
      cmm_diag =
      cmm_cmc_filter(TArgs->cmcFilter,TArgs->
      cmcChangeList,TArgs->cmcChangeCount);
      if (cmm_diag != CMM_OK) {
    fprintf(stderr,"%d: error during
  filter change CR:%d\n",pthread_self(),cmm_diag);
    fprintf(stderr,"Details: %s\n",cmm_strerror(cmm_diag));
    cmm_cmc_unregister();
    pthread_exit((void*)NULL);
      }
      TArgs->FilterChange = 0;
    }

    pthread_testcancel();
  }

  /*NOTREACHED*/

  pthread_cleanup_pop(0);


  /*NOTREACHED*/
  pthread_exit((void*)NULL);

}

/*callback activated when event received*/

void notification_thread_cb
(const cmm_cmc_notification_t * change_notification,void *client_data) {

  printf("%d: notif thread received event (%d) for node %d\n",
     pthread_self(),
     change_notification->cmchange,
     change_notification->nodeid);
}




int main(int argc , char ** argv) {
  int diag;
  pthread_attr_t attr;
  cmm_error_t cmm_diag;
  pthread_t NotificationThread;
  thrArg_t targ;


  /*initialize notification thread*/
  diag = pthread_attr_init(&attr);
  if (diag != 0) {
```

```
  fprintf(stderr,"%d: attr init error\n",pthread_self());
  exit(1);
}

diag = pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_JOINABLE);
if (diag != 0) {
  fprintf(stderr,"%d:
  pthread_attr_setdetachstate error\n",pthread_self());
  exit(1);
}


/*clear event filter*/
targ.cmcFilter = CMM_CMC_NOTIFY_NONE;
targ.cmcChangeCount = 0;

cmm_diag =
cmm_cmc_filter(targ.cmcFilter,targ.cmcChangeList,targ.cmcChangeCount);
if (cmm_diag != CMM_OK) {
  fprintf(stderr,"%d: error during
  filter change CR:%d\n",pthread_self(),cmm_diag);
  fprintf(stderr,"Details: %s\n",cmm_strerror(cmm_diag));
  exit(1);
}

targ.FilterChange = 0;





/*add "vice-master elected" event*/
targ.cmcFilter      = CMM_CMC_NOTIFY_ADD;
targ.cmcChangeCount = 1;
targ.cmcChangeList[0]  = CMM_VICEMASTER_ELECTED;

cmm_diag =
cmm_cmc_filter(targ.cmcFilter,targ.cmcChangeList,targ.cmcChangeCount);
if (cmm_diag != CMM_OK) {
  fprintf(stderr,"%d: error during
  filter change CR:%d\n",pthread_self(),cmm_diag);
  fprintf(stderr,"Details: %s\n",cmm_strerror(cmm_diag));
  exit(1);
}


/*launch notification thread*/

diag = pthread_create
(&NotificationThread,&attr,notification_thread,(void*)&targ);
if (diag != 0) {
  fprintf(stderr,"pthread_create error\n");
  exit(1);
  }

  /*
   * ............
   */


  sleep(20);


  /*We can only register once, tha same thread has to be used*/
  /*to catch all events*/

  /*add CMM_MASTER_DEMOTED and CMM_MASTER_ELECTED events*/

  printf("%d: adding CMM_MASTER_ELECTED and
   CMM_MASTER_DEMOTED events..\n",pthread_self());

  targ.cmcFilter         = CMM_CMC_NOTIFY_ADD;
  targ.cmcChangeCount    = 2;
  targ.cmcChangeList[0] = CMM_MASTER_ELECTED;
  targ.cmcChangeList[1] = CMM_MASTER_DEMOTED;

  /*ask the notification thread to update the filter*/
  /*we don't want to stop it*/
  targ.FilterChange = 1;

  sleep(20);

  /*stop the notification thread*/

  pthread_cancel(NotificationThread);
  pthread_join(NotificationThread,NULL);

  exit (0);

}
```

Index