High Availability Cluster
Multi-Processing for AIX

Programming Client Applications

Version 5.4.1

SC23-4865-10

Tenth Edition (October 2007)

Before using the information in this book, read the general information in Notices for HACMP Programming Client Applications.

This edition applies to HACMP for AIX, v. 5.4.1 and to all subsequent releases of this product until otherwise indicated in new editions.

Contents

About This Guide

This guide describes the Cluster Information Program (Clinfo) client application programming interfaces (APIs) and the Management Information Base (MIB) supplied with the AIX High Availability Cluster Multi-Processing v.5.3 software.

Applications can access information about an HACMP cluster that is stored in the HACMP for AIX MIB. They can do this directly by making Simple Network Management Protocol (SNMP) requests, or indirectly by using the Clinfo C or C++ APIs.

The following table provides version and manual part numbers for *Programming Client Applications*.

HACMP Version	Book Name	Book Number
5.4.1	*Programming Client Applications*	SC23-4865-10
5.4	*Programming Client Applications*	SC23-4865-09
5.3 update 7/2006	*Programming Client Applications*	SC23-4865-08
5.3 update 8/2005	*Programming Client Applications*	SC23-4865-07
5.3	*Programming Client Applications*	SC23-4865-06
5.2 last update 10/2005	*Programming Client Applications*	SC23-4865-05
5.1 last update 6/2004	*Programming Client Applications*	SC23-4865-02

Who Should Use This Guide

This guide is intended for application developers who want to write highly available applications that run in an HACMP for AIX clustered environment or applications that want information about the cluster components. The guide assumes the reader knows the C or C++ programming language and is familiar with networking concepts.

README File

For additional information about the Clinfo API(s), see the **/usr/es/sbin/cluster/samples/clinfo/README** file that accompanies the product.

Highlighting

This guide uses the following highlighting conventions:

Italic	Identifies new terms or concepts, or indicates emphasis.
Bold	Identifies routines, commands, keywords, files, directories, menu items, and other items whose actual names are predefined by the system.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of program code similar to what you might write as a programmer, messages from the system, or information that you should actually type.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

HACMP Publications

The HACMP software comes with the following publications:

- *HACMP for AIX Release Notes* in **/usr/es/sbin/cluster/release_notes** describe issues relevant to HACMP on the AIX platform: latest hardware and software requirements, last-minute information on installation, product usage, and known issues.
- *HACMP on Linux Release Notes* in **/usr/es/sbin/cluster/release_notes.linux/** describe issues relevant to HACMP on the Linux platform: latest hardware and software requirements, last-minute information on installation, product usage, and known issues.
- *HACMP for AIX: Administration Guide,* SC23-4862
- *HACMP for AIX: Concepts and Facilities Guide,* SC23-4864
- *HACMP for AIX: Installation Guide,* SC23-5209
- *HACMP for AIX: Master Glossary,* SC23-4867
- *HACMP for AIX: Planning Guide,* SC23-4861
- *HACMP for AIX: Programming Client Applications*, SC23-4865
- *HACMP for AIX: Troubleshooting Guide,* SC23-5177
- *HACMP on Linux: Installation and Administration Guide,* SC23-5211
- *HACMP for AIX: Smart Assist Developer's Guide,* SC23-5210
- *IBM International Program License Agreement.*

HACMP/XD Publications

The HACMP Extended Distance (HACMP/XD) software solutions for disaster recovery, added to the base HACMP software, enable a cluster to operate over extended distances at two sites. HACMP/XD publications include the following:

- *HACMP/XD for Geographic LVM (GLVM): Planning and Administration Guide,* SA23-1338
- *HACMP/XD for HAGEO Technology: Concepts and Facilities Guide, SC23-1922*
- *HACMP/XD for HAGEO Technology: Planning and Administration Guide,* SC23-1886
- *HACMP/XD for Metro Mirror: Planning and Administration Guide,* SC23-4863.

HACMP Smart Assist Publications

The HACMP Smart Assist software helps you quickly add an instance of certain applications to your HACMP configuration so that HACMP can manage their availability. The HACMP Smart Assist publications include the following:

- *HACMP Smart Assist for DB2 User's Guide, SC23-5179*
- *HACMP Smart Assist for Oracle User's Guide, SC23-5178*
- *HACMP Smart Assist for WebSphere User's Guide, SC23-4877*
- *HACMP for AIX: Smart Assist Developer's Guide, SC23-5210*
- *HACMP Smart Assist Release Notes.*

IBM AIX Publications

The following publications offer more information about IBM technology related to or used by HACMP:

- *RS/6000 SP High Availability Infrastructure*, SG24-4838
- *IBM AIX v.5.3 Security Guide,* SC23-4907
- *IBM Reliable Scalable Cluster Technology for AIX and Linux: Group Services Programming Guide and Reference,* SA22-7888
- *IBM Reliable Scalable Cluster Technology for AIX and Linux: Administration Guide,* SA22-7889
- *IBM Reliable Scalable Cluster Technology for AIX: Technical Reference*, SA22-7890
- *IBM Reliable Scalable Cluster Technology for AIX: Messages*, GA22-7891.

Accessing Publications

Use the following Internet URLs to access online libraries of documentation:

AIX, IBM eServer Series p™, and related products:

http://www.ibm.com/servers/aix/library

AIX v.5.3 publications:

http://www.ibm.com/servers/eserver/pseries/library/

WebSphere Application Server publications:

Search the IBM website to access the WebSphere Application Server Library

DB2 Universal Database Enterprise Server Edition publications:

http://www.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/v8pubs.d2w/en_main#V8PDF

Tivoli Directory Server publications:

http://publib.boulder.ibm.com/tividd/td/IBMDirectoryServer5.1.html

IBM Welcomes Your Comments

You can send any comments via e-mail to hafeedbk@us.ibm.com. Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States or other countries:

- AFS
- AIX
- DFS
- eserver
- eServer Cluster 1600
- Enterprise Storage Server
- HACMP
- IBM
- NetView
- RS/6000
- Scalable POWERParallel Systems
- Series p
- Series x
- Shark
- SP
- WebSphere
- Red Hat Enterprise Linux (RHEL)

Chapter 1: Cluster Information Program

This chapter provides an overview of the Cluster Information Program (Clinfo) and a description of the status information that Clinfo receives and maintains about an HACMP for AIX cluster.

Cluster Information Program Overview

An HACMP cluster can undergo various transitions in its state over time. For example, a node can join or leave the cluster or an application can fallover to a backup node. Each of these changes affects the state of the cluster. Because a cluster is dynamic, an application must be able to obtain current, accurate information about the cluster so that it can respond to changes as they occur. Clinfo provides this service.

The HACMP cluster software exports state information and cluster events through SNMP, an industry standard network protocol. Writing programs to the SNMP API can be non-trivial and may require several transactions to obtain all the information related to a single cluster entity like a node or resource group.

The Clinfo API and associated components provide a library of access routines that supply the same cluster information using a straightforward programming model. For more information about the SNMP information available from HACMP or details of the Clinfo API implementation, please refer to Appendix A: Implementation Specifics.

The following sections provide additional details on using Clinfo in the HACMP environment:

- Getting Started
- Clinfo APIs
- Events Tracked by Clinfo
- Cluster Information Tracked by Clinfo.

Getting Started

The Clinfo API and its associated components are installed and configured when you install HACMP. There are a few things to consider before you begin using Clinfo:

1. In order to use Clinfo you must specify that the Clinfo agent be started when you start HACMP cluster services. This is the **Startup cluster information Daemon** option in `smitty clstart`.
2. Clinfo will recognize and use SNMP community names other than public. If your installation is using a non-public name or if you want to specify a specific name, see Appendix A: Implementation Specifics.

3. Clinfo will run on each node where you install HACMP. It can also run on other machines provided they have TCP/IP connectivity to one of the HACMP nodes. For additional information about this capability, see Appendix A: Implementation Specifics.

Clinfo APIs

An application accesses cluster information through the Clinfo API functions. Developers can use either the Clinfo C API or the Clinfo C++ API to access the cluster status information, which is then available locally for clients running the Clinfo program.

HACMP for AIX includes two versions of the Clinfo C and C++ API libraries: one for single-threaded (non-threaded) applications (**libcl.a** and **libclpp.a**) and one for multi-threaded applications (**libcl_r.a** and **libclpp_r.a**). Be sure to link with the appropriate version for your application. The **libcl_r.a** is a thread-safe version of the **libcl.a**; the **libclpp_r.a** is a thread-safe version of the **libclpp.a**.

Each of these libraries contain 32-bit and 64-bit objects, which are loaded at runtime depending on the AIX operating environment.

See Chapter 2: Clinfo C API and Chapter 3: Clinfo C++ API, for detailed descriptions of the routines in these APIs.

Events Tracked by Clinfo

Clinfo receives status information about the cluster events from the Cluster Manager. This information is accessible by the routines in the APIs. Clinfo tracks topology events, as the cluster passes through various states. Some of the states Clinfo tracks include:

- Cluster state is up or down
- Cluster substate has become stable or unstable
- Application has come online or failed over to a backup node
- Network has failed
- Node is in the process of joining the cluster
- Node has completed joining the cluster
- Node is leaving the cluster (that is, the node has failed)
- Node has left the cluster
- New primary Cluster Manager has been elected (optional event)
- Change of state for a cluster site (if configured).

A complete listing of events can be found in subsequent sections of this document or in the **clinfo.h** include file, which is compiled into your application.

Clinfo receives dynamic reconfiguration events but does not track them; that is, applications cannot register to receive notification of dynamic reconfiguration events. Clinfo sets the cluster substate to CLSS_RECONFIG when it receives dynamic reconfiguration events. Applications can obtain this information using the **cl_getcluster** routine. The events triggered by the dynamic reconfiguration, such as a node up or node down event, are visible to applications.

Cluster Information Tracked by Clinfo

Clinfo maintains information about the following entities:

- Clusters
- Nodes in the clusters
- Network Interfaces attached to each node
- Available Network Interface Information
- State and location of Resource Group
- Cluster Networks
- Cluster Site(s).

Clusters

An HACMP cluster is a group of processors that cooperate to provide a highly available environment.

Available Cluster Information

Clinfo maintains the following information about configured clusters:

- Cluster Name
- Cluster ID
- Cluster State
- Cluster Substate
- Primary Node Name (note that the Primary node is an HACMP concept and does necessarily correspond to the role of the node with respect to the application in use)
- Cluster Number of Nodes
- Cluster Number of Networks
- Cluster Number of Resource Groups
- Cluster Number of Sites.

Cluster Name

A cluster name uniquely identifies a cluster. The administrator specifies the name when the cluster is configured.

Cluster ID

A cluster ID identifies each cluster. The cluster ID is a numeric value assigned by HACMP (this is not user-defined) at the time the cluster is configured.

Cluster State

A cluster can be in one of the following defined states:

CLS_UP	At least one node in the cluster is up, and a primary is defined.
CLS_DOWN	No nodes are up.

CLS_UNKNOWN	Clinfo is unable to communicate, or is not yet communicating with an SNMP process on any active cluster.
CLS_ NOTCONFIGURED	The cluster is not yet configured.

Cluster Substate

A cluster can be in one of several defined substates:

CLSS_ERROR	An error occurred and manual intervention is required.
CLSS_RECONFIG	A dynamic reconfiguration of the cluster is in progress.
CLSS_STABLE	The cluster is stable (no reconfiguration is occurring).
CLSS_UNSTABLE	The cluster is unstable. The event history will show what events are happening.
CLSS_UNKNOWN	Clinfo is unable to communicate with an SNMP process on a cluster node
CLSS_ NOTCONFIGURED	The cluster is not yet configured.
CLSS_NOTSYNCHED	Some basic cluster configuration information exists, but the cluster has not yet been verified and synchronized.

Primary Node Name

The name of the node elected primary by its peers. This is an optional function carried over from a previous version of the software.

Cluster Number of Nodes

The number of nodes defined in the cluster.

Cluster Number of Networks

The total number of networks in the cluster.

Cluster Number of Resource Groups

The total number of resource groups configured.

Cluster Number of Sites

If sites are in use, the number of sites configured.

Nodes

A node is one of the processors that make up the cluster. Each node in the cluster runs the **clstrmgr**, **clinfo**, and **clsmuxpd** daemons.

Available Node Information

Clinfo maintains the following information about a node:

- Cluster ID
- Node Name

- Node State
- Network Interfaces (service, standby, and tty).

Cluster ID

The ID of the cluster to which this node belongs.

Node Name

The node name is a user-assigned string. The node name can contain up to 32 characters, and cannot begin with a leading numeric.

Node State

A node can be in one of following defined states:

CLS_UP	The node is up and running
CLS_DOWN	The node is down
CLS_JOINING	The node is in the process of joining the cluster
CLS_LEAVING	The node is in the process of leaving the cluster.

Network Interfaces

The number and addresses of service interfaces attached to the node.

Network Interfaces

A network interface is the physical connection between a node and a network.

An HACMP cluster can support multiple networks and point-to-point connections, as well as an RS232 serial line point-to-point connection. Each network will have one or more network interfaces on each cluster node.

Available Network Interface Information

Clinfo maintains the following information about a network interface:

- Cluster ID
- Node Name
- Active Node ID
- Interface Name
- Interface ID
- Interface Address
- Interface State
- Interface Role.

Cluster ID

The ID of the cluster to which this interface belongs.

Node Name

The name of the node to which this interface is attached.

Active Node ID
ID of the node where the address is currently active.

Interface Name
An interface's name is the same as the name in the **/etc/hosts** file for the interface (that is, the name associated with the IP address of the host).

Interface ID
The network ID of the network to which this interface is connected.

Interface Address
The IP address for the interface as defined in the **/etc/hosts** file.

Interface State
An interface can be in one of several defined states. The following values describe the state of a network interface:

CLS_UP	The interface is up and running.
CLS_DOWN	The network interface or network is down.
CLS_INVALID	This interface is not defined for this node.

Interface Role
An interface can be in one of several defined roles. The following values describe the roles of a network interface:

CL_INT_ROLE_INVALID	The network interface is invalid.
CL_INT_ROLE_SERVICE	The network interface is defined as a service interface.
CL_INT_ROLE_STANDBY	The network interface role is deprecated.
CL_INT_ROLE_BOOT	The network interface is defined as a service interface.
CL_INT_ROLE_SH_SERVICE	The network interface role is deprecated.

Resource Group

A resource group contains all the resources associated with an instance of an application that HACMP is keeping highly available.

Resource groups are brought online as nodes join the cluster. Resource groups are moved between cluster nodes as failures occur. The groups' state and location are available through Clinfo.

Available Resource Group Information

Clinfo maintains the following information about a resource group:

- Cluster ID
- Group Name
- Group ID

- Group Startup Policy
- Group Fallover Policy
- Group Fallback Policy
- Group Site Policy
- Number of Nodes
- Group Node IDs
- Group Node State.

Cluster ID

The ID of the cluster to which this resource group belongs.

Group Name

This is the name given to the resource group when it is first defined.

Group ID

The numeric ID associated with the group.

Group Startup Policy

The startup policy used for this resource group:

- Online on home node only
- Online on first available node
- Online on all available nodes
- Online using distribution policy.

Group Fallover Policy

The fallover policy for this resource group:

- Fall over to the next priority node in the list
- Fall over using dynamic node priority
- Bring offline (or error node only).

Group Fallback Policy

The fallback policy for this resource group:

- Fall back to a higher priority node in the list
- Never fall back.

Group Site Policy

When the group contains replicated resources, the following policies are used for resource group startup, fallover, and fallback as it occurs between two sites:

- Prefer primary site
- Online on either site
- Online on both sites.

Number of Nodes

The number of nodes participating in the resource group.

Group Node IDs

The node ID of all nodes participating in the resource group.

Group Node State

The state of the resource group on each node.

Resource Group States

Resource groups can be in one of the following states on one or more cluster nodes:

CL_RGNS_INVALID	The node is not part of this resource group.
CL_RGNS_ONLINE	The group and all its resources are up and available on the node.
CL_RGNS_OFFLINE	The group is currently not active on the node.
CL_RGNS_ACQUIRING	The group is in the process of being acquired on this node.
CL_RGNS_RELEASING	The group is being released from this node.
CL_RGNS_ERROR	An error occurred when trying to bring the group online or offline. Resources for this group are not active. Manual intervention is required.

Note: This list does not include all possible resource group states: if sites are defined, a primary and a secondary instance of the resource group could be online, offline, in the error state, or unmanaged. In addition, the resource group instances could be in the process of acquiring or releasing. The corresponding resource group states are not listed here, but have descriptive names that explain which actions take place.

Cluster Networks

An HACMP cluster may contain both TCP/IP and non-IP based networks. Typically, non-IP based networks are used for disk heartbeating traffic.

Available Network Information

Clinfo provides the following information about the networks in the cluster:

- Network Name
- Network ID
- Network Type
- Network Attribute
- Network Node Information
- Network State.

Network Name

The name associated with the network. You can supply this name or HACMP can generate it.

Network ID

A numeric network identifier generated by HACMP.

Network Type

The physical type of the network, such as ether, token, or hps.

Network Attribute

The network attribute can be one of the following:

Serial	The network is non-IP based, such as RS232, target mode, or disk heartbeating.
Public or **Private**	For IP based networks, this attribute can be set to Public or Private. Setting the attribute to Private identifies this network for use by Oracle as a private network. The default is Public.

Network Node Information

For each cluster node connected to the network, the following information is provided:

Node ID	The node ID of each node.
Node State	The state of the network on the node. Note that this may be different than the *global* network state.

Network State

The network state, including the global state and the per node state, can be one of several predefined values:

CLS_UP	The network is up. If the network is up on any node, the global state will be set to up as well.
CLS_DOWN	The network is not up. A network can be down on one or more nodes (that is the per node state is **CLS_DOWN**) and still have a global state of **CLS_UP**.

Cluster Site(s)

When sites are configured, nodes in the cluster are grouped into cluster sites. The state of the cluster sites is tracked and cluster events are generated.

Available Cluster Site Information

Clinfo maintains the following information about a cluster site:

- Site ID
- Site Name
- Site Priority
- Site Backup Method
- Site State
- Site Number of nodes
- Site Node IDs.

Site ID

The HACMP generated ID for the site.

Site Name

The site name specified when the site was created.

Site Priority

Site priority determines the precedence and direction for data mirroring between sites. A cluster site can have a priority of one of the following:

CL_SITE_PRIMARY	This is the site where the application runs. Data is mirrored from this site to the backup site.
CL_SITE_SECONDARY	This site serves as a backup.
CL_SITE_TERTIARY	This site is mirroring data sent from the secondary site. This value is reserved for future use.

Site Backup Method

This is the backup communications method for the site. If a backup method is configured, it can be one of the following:

CL_SITE_BACKUP_DBFS	Dial Back Fail Safe
CL_SITE_BACKUP_SGN	Serial Global Network
CL_SITE_BACKUP_NONE	No backup communications are configured

Site State

The global state of the site:

CLS_UP	One or more node(s) in the site is up
CLS_DOWN	All the nodes in the site are down

Site Number of nodes

The number of nodes in this site.

Site Node IDs

A list of node IDs that participate in this site.

Chapter 2: Clinfo C API

The Clinfo C Application Programming Interface (API) is a high-level interface that you can use in an application to get status information about an HACMP cluster. This chapter describes the specific C language routines and utilities available in the Clinfo C API. It contains the following sections:

- Using the Clinfo C API in an Application
- Upgrading Applications from Earlier Clinfo Releases
- Memory Allocation Routines
- Requests
- Utilities
- Routines.

Note: HACMP 5.3 eliminated **cl_registerwithclsmuxpd()** API. Any application compiled with this API will fail to load. Use application monitoring instead of the **cl_registerwithclsmuxpd()** routine. See the section on Application Monitoring in Chapter 2: Initial Cluster Planning in the *Planning Guide*.

Before reading this chapter, you should read Chapter 1: Cluster Information Program, which describes the types of information Clinfo maintains about an HACMP cluster.

Using the Clinfo C API in an Application

This section describes how to use the Clinfo C API in an application.

HACMP for AIX includes separate libraries for multi-threaded and for single-threaded applications. Be sure to link with the appropriate library for your application.

Note: The Clinfo **cluster.es.client.lib** library contains the **libcl.a** with both 32- and 64-bit objects. You must recompile/relink your application in a 64-bit environment to get a 64-bit application using the Clinfo API.

Header Files

You must specify the following **include** directives in each source module that uses the Clinfo C API:

```
#include <sys/types.h>
#include <netinet/in.h>
#include <cluster/clinfo.h>
```

In addition to this list of **include** directives, specify the following **include** directive in each source module that uses the **cl_registereventnotify** routine:

```
#include <signal.h>
```

Compiler Preprocessor Directives for Applications Using Client APIs

When compiling applications using client APIs, use the compiler flag **--D__HAES__**. This is required because of the **#ifdef** preprocessor directives used in the header files. This is due to previous support of another version of HACMP (HAS).

Linking the libcl.a and libcl_r.a Libraries

You must add the following directives to the object load command of a *single-threaded* application that uses the Clinfo C API:

```
-lcl
```

You must add the following directives to the object load command of a *multi-threaded* application that uses the Clinfo C API:

```
-lcl_r
```

The **libcl.a** and **libcl_r.a** libraries contain the routines that support the Clinfo C API.

Constants

The Clinfo C API routines use the following constants, defined in the **clinfo.h** file:

CL_MAXNAMELEN	The maximum length of a character string naming a cluster, node, or interface (256). The value of **CL_MAXNAMELEN** is the same as **MAXHOSTNAMELEN**, defined in the **sys/param.h** file.
CL_MAX_EN_REQS	The maximum number of event notification requests allowed (10).
CL_ERRMSG_LEN	Maximum error message length (128 characters).

Macros

HACMP 5.2 replaced the constants CL_MAXCLUSTERS, CL_MAXNODES, and CL_MAXNETIFS with macros that provide the current sizes of the various arrays in the data structures. The macros have the same name as the constants they replace. You cannot use these macros as array bounds in declaration statements.

CL_MAXCLUSTERS	Provides current size of space in data structures for the array holding information about the number of clusters.
CL_MAXNODES	Provides current size of space in data structures for the array holding information about the number of nodes in a cluster.
CL_MAXNETIFS	Provides current size of space in data structures for the array holding information about the number of interfaces attached to a node.
CL_MAXGROUPS	Provides current size of space in data structures for the array holding information about the number of resource groups.

Data Types and Structures

The Clinfo C API uses the following data types and structures, defined in the **clinfo.h** file.

- Enumerated Type Containing State Information
- Enumerated Type Containing Substate Information
- Enumerated Type Containing Resource Group State
- Enumerated Type Containing Resource Group Policies
- Enumerated Type Containing Interface Roles
- Enumerated Type Containing Network Attribute
- Enumerated Type Containing Site Priority
- Enumerated Type Containing Site Backup Communication Methods
- Data Structure Representing a Resource Group
- Data Structure Representing a Network Interface
- Data Structure Representing a Node
- Data Structure Representing a Cluster
- Data Structure Representing an Event Notification Registration Request
- Data Structure Representing an Event Notification Message
- Data Structure Representing a Cluster Network
- Data Structure Representing a Cluster Site.

Enumerated Type Containing State Information

The following enumerated data type describes the state of a cluster, node, interface, or event notification:

```
enum cls_state {
    CLS_INVALID,
    CLS_VALID,
    CLS_UP,
    CLS_DOWN,
    CLS_UNKNOWN,
    CLS_GRACE,
    CLS_JOINING,
    CLS_LEAVING,
    CLS_IN_USE,
    CLS_PRIMARY
};
```

Enumerated Type Containing Substate Information

The following enumerated data type describes the substate of a cluster:

```
enum cls_substate {
    CLSS_UNKNOWN,
    CLSS_STABLE,
    CLSS_UNSTABLE,
    CLSS_ERROR,
    CLSS_RECONFIG
    CLSS_NOT_CONFIGURED
};
```

Enumerated Type Containing Resource Group State

The following enumerated data type contains all states a resource group can be in on a node:

```
enum cl_resource_states{
    CL_RGNS_INVALID=1,
    CL_RGNS_ONLINE=2,
    CL_RGNS_OFFLINE=4,
    CL_RGNS_ACQUIRING=16,
    CL_RGNS_RELEASING=32,
    CL_RGNS_ERROR=64
};
```

Enumerated Type Containing Resource Group Policies

The following enumerated data type contains all resource group policies (both node and site):

```
enum cl_rg_policies {
    CL_RGP_INVALID = 0,
    CL_RGP_ONLINE_ON_HOME_NODE = 1,
    CL_RGP_ONLINE_ONFIRST_AVAILBLE_NODE = 2,
    CL_RGP_ONLINE_USING_DISTRIBUTION_POLICY = 3,
    CL_RGP_ONLINE_ALL_NODES = 4,
    CL_RGP_FALLOVER_TO_PRIORITY_NODE = 5,
    CL_RGP_FALLOVER_USING_DNP = 6,
    CL_RGP_BRING_OFFLINE = 7,
    CL_RGP_FALLBACK_TO_HIGHER_PRIORITY_NODE = 8,
    CL_RGP_NEVER_FALLBACK = 9,
    CL_RGP_PREFER_PRIMARY_SITE = 10,
    CL_RGP_ONLINE_ON_EITHER_SITE = 11,
    CL_RGP_ONLINE_ON_BOTH_SITES = 12,
    CL_RGP_IGNORE_SITES = 13
};
```

Enumerated Type Containing Interface Roles

```
enum cl_interface_role {
    CL_INT_ROLE_INVALID = 0,
    CL_INT_ROLE_SERVICE = 16,
    CL_INT_ROLE_STANDBY = 32,      /* deprecated */
    CL_INT_ROLE_BOOT = 64,
    CL_INT_ROLE_SH_SERVICE = 128,  /* deprecated */
};
```

Enumerated Type Containing Network Attribute

The following enumerated data type specifies the attributes for a network:

```
/*
* Enumeration of Network attributes. Note that the public/private
attribute is for use by Oracle only - HACMP does not use this attribute.
*/

enum cl_network_attribute
{
    CL_NET_ATTR_INVALID = 0, /* IP networks can be public or private */
    CL_NET_TYPE_PUBLIC = 1,
    CL_NET_TYPE_PRIVATE = 2, /* non-IP (serial) networks are always */
    CL_NET_TYPE_SERIAL = 4
};
```

Enumerated Type Containing Site Priority

The following enumerated data type describes the priority for a site. Sites are defined as having a priority when processing resources.

```
/*
* Enumeration of Site Priorities
*/

enum cl_site_priority
{
    CL_SITE_PRI_NONE = 0,
    CL_SITE_PRI_PRIMARY = 1,
    CL_SITE_PRI_SECONDARY = 2,
    CL_SITE_PRI_TERTIARY = 4
};
```

Enumerated Type Containing Site Backup Communication Methods

The following enumerated data type contains optional backup methods that can be configured for a site:

```
/*
* Enumeration of Site Backup Communication Options
*/

enum cl_site_backup
{
    CL_SITE_BACKUP_NONE = 0,
    CL_SITE_BACKUP_DBFS = 1,
    CL_SITE_BACKUP_SGN = 2
};
```

Data Structure Representing a Resource Group

The following structure contains all information available for each resource group:

```
struct cl_group {
    int  clg_clusterid;
    int  clg_group_id;
    char clg_name[CL_MAXNAMELEN];
    enum cl_rg_policies clg_policy;          /* deprecated */
    enum cl_rg_policies clg_startup_policy;
    enum cl_rg_policies clg_fallover_policy;
    enum cl_rg_policies clg_fallback_policy;
    enum cl_rg_policies clg_site_policy;
    char clg_user_policy_name[CL_MAXNAMELEN];
    int  clg_num_nodes;
    int  clg_node_ids[MAXNODES];
        /* list of nodes (ids) in this group */
    enum cl_resource_states clg_node_states[MAXNODES];
        /* state on each */
    int  clg_num_resources;
    int  clg_resource_id[MAXRESOURCES];
        /* list of resources (id) group */
    enum cl_resource_states clg_res_state[MAXRESOURCES]; /* state
       */ int clg_vrmf;                      /* version of this client
*/
};
```

Data Structure Representing a Network Interface

The following data structure represents a network interface:

```
struct cl_netif {
    int cli_clusterid;              /* Cluster Id */
    int cli_nodeid;                 /* Cluster node Id - used internally only */
    char cli_nodename[CL_MAXNAMELEN];/* Cluster node name */
    int cli_interfaceid;            /* Cluster Node Interface Id */
    enum cls_state cli_state;       /* Cluster Node Interface State */
    char cli_name[CL_MAXNAMELEN];   /* Cluster Node Interface Name */
    struct sockaddr_in cli_addr;    /* Cluster Node Interface IP Address */
    int cli_active_nodeid;          /* Cluster node Id where addr is up */
    enum cl_interface_role cli_role; /* Role of interface (boot/service)*/
    int cli_networkid;              /* Cluster Network ID for this Interface */
    int cli_vrmf;
};
```

Data Structure Representing a Node

The following data structure represents a cluster node:

```
struct cl_node {
    int cln_clusterid;                  /* Cluster Id */
    int cln_nodeid;                     /* Cluster node Id */
    char cln_nodename[CL_MAXNAMELEN];   /* Cluster node name */
    enum cls_state cln_state;           /* node state */
    int cln_nif;                        /* number of interfaces */
    struct cl_netif *cln_if;            /* interfaces */
    int cln_glidle;                     /* CPU.glidle */
    int cln_real_mem_free;              /* Paging space utilitization */
    int cln_disk_busy;                  /* disk busy */
    int cln_vrmf;                       /* version of this client */
};
```

Data Structure Representing a Cluster

The following data structure represents a cluster:

```
struct cl_cluster{
    int  clc_clusterid;                /* cluster id*/
    enum cls_state clc_state;          /* Cluster State */
    enum cls_substate clc_substate;    /* Cluster Substate */
    char clc_primary[CL_MAXNAMELEN];   /* Cluster Primary Node */
    char clc_name[CL_MAXNAMELEN];      /* Cluster Name */
    int  clc_number_of_nodes;      /* number of cluster nodes */
    int  clc_number_of_groups;      /* number of resource groups */
    int  clc_number_of_networks;       /* number of networks */
    int  clc_number_of_sites;      /* number of sites */
    int clc_vrmf;      /* version of this client */
};
```

Data Structure Representing an Event Notification Registration Request

The following data structure represents an event notification registration request:

```
struct cli_enr_req_t {
    int event_id;                  /* event id */
    int cluster_id;                /* cluster id */
    int node_id;                   /* node id(internal use only)*/
    char node_name[CL_MAXNAMELEN]; /* node name */
    int net_id;                    /* network id */
    int signal_id;                 /* signal id */
    int vrmf;
};
```

Data Structure Representing an Event Notification Message

The following data structure represents an event notification message:

```
struct cli_en_msg_t {
    int event_id;                  /* event id */
    int cluster_id;                /* cluster id */
    int node_id;                   /* node id(internal use only)*/
    char node_name[CL_MAXNAMELEN]; /* node name */
    int net_id;                    /* network id */
    int vrmf;
};
```

Data Structure Representing a Cluster Network

The following data structure contains information for each cluster network:

```
/*
* Structure containing information relating to a network.
*/
struct cl_net {
    int clnet_clusterid;                   /* Cluster Id */
    char clnet_name[CL_MAXNAMELEN];        /* Cluster network name */
    int clnet_id;                          /* Cluster Network Id */
    char clnet_type[CL_MAXNAMELEN];        /* ether, token, etc */
    enum cl_network_attribute clnet_attr;  /* public/serial */
    enum cls_state clnet_state;            /* Cluster Network State */
        /* Note that this is the cluster wide or "global" network state */
        /* which may be different than the state of the network on any */
        /* particular node */
    int clnet_numnodes;
        /* Number of nodes connected to this Network */
    int clnet_node_ids[MAXNODES];
        /* Node ids connected to this Network */
    enum cls_state clnet_node_states[MAXNODES];
        /* Network State per Node */
    int clnet_vrmf;  /* version of this client */
};
```

Data Structure Representing a Cluster Site

The following data structure contains information for each site configured in an HACMP cluster. Note that site configuration is optional.

```
/*
* Structure containing information relating to a site
*/
struct cl_site {
    int clsite_clusterid;                  /* Cluster Id */
    int clsite_id;
    char clsite_name[CL_MAXNAMELEN];
    enum cl_site_priority clsite_priority;
    enum cl_site_backup clsite_backup;
    enum cls_state clsite_state;           /* Cluster Site State */
    int clsite_numnodes;
    int clsite_nodeids[MAXNODES];          /* List of nodes (ids) in this group */
    int clsite_vrmf;                       /* version of this client */
};
```

Upgrading Applications from Earlier Clinfo Releases

In prior releases of HACMP, the cluster ID was configured manually; it is now created automatically. There are several calls that will return the cluster ID, given the cluster name or other parameters:

- **cl_getclusterid**—given a cluster name, returns cluster ID
- **cl_getclusteridbyifaddr**—given a network interface address, returns cluster ID
- **cl_getclusteridbyifname—**given a network interface name, returns cluster ID.

If your application uses API calls that require the cluster ID, you must add a call to one of these routines that return the cluster ID.

Earlier releases of Clinfo used integers to identify cluster nodes (node IDs) instead of character strings (node names).

The following sections give you some examples of how to convert calls in your application to various Clinfo C API routines that previously used a *nodeid* parameter. For each routine, an example of its use with node IDs is followed by an example using node names.

cl_getlocalid

The following is an example of the **cl_getlocalid** routine from an earlier release that uses node ID:

```
int clusterid, nodeid, status;
status = cl_getlocalid (&clusterid, &nodeid);
if (status != CLE_OK) {
    if (status == CLE_IVNODE) {
        printf ("This node is not a cluster member");
    } else {
        cl_perror (status, "Can't get local cluster ID");
        }
} else {
    printf ("member of cluster %d node %d",clusterid, nodeid);
}
```

In the new version of the example of the C API **cl_getlocalid** routine, note the change in the declaration statement. Where previously all variables were declared as int, now you must declare *nodename* as a character string and make the corresponding changes to the **printf** statements.

```
int clusterid, status;
char nodename[CL_MAXNAMELEN];
status = cl_getlocalid (&clusterid, nodename);
if (status != CLE_OK) {
    if (status == CLE_IVNODE) {
        printf ("This node is not a cluster member");
    } else {
        cl_perror (status, "Can't get local cluster ID");
    }
} else {
    printf ("member of cluster %d node %s",clusterid, nodename);
}
```

cl_getnodeidbyifname

The following is an example of the **cl_getnodeidbyifname** routine from an earlier release that uses node ID:

```
int clusterid, nodeid;
char *interfacename;
strcpy (interfacename,"editserver");
clusterid = 1;
nodeid = cl_getnodeidbyifname (clusterid, interfacename);
if (nodeid < 0) {
    cl_perror(nodeid,"Can't get node ID");
} else {
    printf("ID of %s on cluster %d is %d", interfacename, clusterid, nodeid);
}
```

In the new version of the example of the C API **cl_getnodeidbyifname** routine, note that the name of the routine itself has changed to **cl_getnodenamebyifname**. You must change the declaration to include *nodename* as a string, instead of *nodeid* as an int; then make the corresponding changes to the **printf** statements.

```
int clusterid, status;
char nodename[MAXNAMELEN];
char *interfacename[MAXNAMELEN];
strcpy (interfacename,"editserver");
clusterid = 1;
status = cl_getnodenamebyifname (clusterid, interfacename, nodename);
if (status != CLE_OK)
    cl_perror(nodename,"Can't get node name");
} else {
    printf("name of %s on cluster %d is %s", interfacename, clusterid, nodename);
}
```

cl_getprimary

Here is an example of the **cl_getprimary** routine, from an earlier release, using node ID:

```
int clusterid, primary; /* clusterid is arbitrary.*/
clusterid = 1;
primary = cl_getprimary (clusterid);
if (primary < 0) {
    cl_perror (primary, "Can't get cluster primary");
} else {
    printf ("Primary node for cluster %d is %d", clusterid, primary);
}
```

In the new version of the C API **cl_getprimary** routine example, note the change in the declaration to *nodename* as a string instead of *nodeid* as an int, and the corresponding changes to the **printf** statements. Moreover, the if statement must be changed since the previous version depended on the fact that the desired information (*nodeid*) was an int; now it is a string (*nodename*).

```
int clusterid, status;
char nodename[CL_MAXNAMELEN];
/* clusterid is arbitrary. */
clusterid = 1;
status = cl_getprimary (clusterid, nodename);
if (status != CLE_OK) {
    cl_perror (status, "Can't get cluster primary");
} else {
    printf ("Primary node for cluster %d is %s", clusterid, nodename);
}
```

cl_isaddravail

The following is an example of the **cl_isaddravail** routine from an earlier release that uses node ID:

```
int clusterid, nodeid, status;
struct sockaddr_in addr;      /* clusterid, nodeid, and addr are
arbitrary.*/
clusterid = nodeid = 1;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr ("1.1.1.1");
status = cl_isaddravail (clusterid, nodeid, &addr);
if (status != CLE_OK) {
    cl_perror (status, "Interface not available");
} else {
    printf ("Interface address for %s is available", inet_ntoa
    (addr.sin_addr.s_addr));
}
```

In the new version of the C API **cl_isaddravail** routine example, the declaration statement changes to use *nodename* instead of *nodeid.* See the corresponding changes to the **printf** statements. Moreover, the assignment statement changes to use strcpy for the *nodename*.

```
int clusterid, status;
char nodename[CL_MAXNAMELEN];
struct sockaddr_in addr;    /* clusterid, nodename, and addr are arbitrary. */
clusterid = 1;
strcpy (nodename, "node1");
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr ("1.1.1.1");
status = cl_isaddravail (clusterid, nodename, &addr);
if (status != CLE_OK) {
    cl_perror (status, "Interface not available");
} else {
    printf ("Interface address for %s is available", inet_ntoa
    (addr.sin_addr.s_addr));
}
```

Memory Allocation Routines

The following routines allocate or free memory used to store cluster or node map information. You must call the appropriate memory allocation routines before the corresponding retrieval routine, and then call the free routine to release the storage when done.

cl_alloc_clustermap	Allocates storage for a list of four clusters of 32 nodes with 32 interfaces each.
cl_alloc_groupmap	Allocates storage for a list of 64 resource groups.
cl_alloc_netmap	Allocates storage for a list of networks.
cl_alloc_nodemap	Allocates storage for a list of 32 nodes with 32 interfaces each.
cl_alloc_sitemap	Allocates storage for a list of sites.
cl_free_clustermap	Frees the storage for a list of clusters that was allocated by calling **cl_alloc_clustermap**.

cl_free_groupmap	Frees storage for a list of resource groups allocated with **cl_alloc_groupmap**.
cl_free_netmap	Frees the storage for a list of networks that was allocated by calling **cl_alloc_netmap**.
cl_free_nodemap	Frees the storage for a list of clusters that was allocated by calling **cl_alloc_nodemap**.
cl_free_sitemap	Frees the storage for a list of sites that was allocated by calling **cl_alloc_sitemap**.

The following example illustrates how to use these routines. Note that you no longer use CL_MAXNODES to allocate storage for the returned information on the nodes in the cluster. For additional examples, see the reference pages for the **cl_getclusters** and **cl_getnodemap** routines.

```
int status;
struct cl_cluster *clustermap;

cl_alloc_clustermap (&clustermap);
status = cl_getclusters(clustermap);
if (status < 0) {
    cl_perror(status, "Can't get cluster information");
} else {
    printf("There are currently %d running clusters", status);
}
...
cl_free_clustermap (clustermap);
```

There is an additional new API **cl_node_free()**, which frees storage associated with a single cl_node struct.

Consider the following example:

```
struct cl_node nodebuf;
    cl_getnode(clusterid, "ppstest5", &nodebuf);
    printf(“Node %s is id %d\n”, nodebuf.cln_nodename,
nodebuf.cln_nodeid);
    cl_node_free(&nodebuf);
);
```

After the call to **cl_getnode()**, the cln_if field is filled in with the list of network interface structs associated with the node. This list is dynamically allocated by **cl_getnode()** and must be freed to avoid a memory leak in a long-running program.

cl_node_free() frees the network interface storage field of the **cl_node** structure. See the subsequent API description for **cl_node_free()** for more information.

Requests

The Clinfo C API has the following types of requests:

- Cluster Information Requests
- Node Information Requests
- Network Interface Information Requests
- Network Information Requests
- Event Notification Requests.
- Resource Group Information Requests
- Site Information Requests.

Cluster Information Requests

The following cluster information requests return information about a cluster:

cl_getcluster	Returns all known information about the cluster with the specified cluster ID.
cl_getclusterid	Returns the cluster ID of the cluster with the specified cluster name.
cl_getclusteridbyifaddr	Returns the cluster ID of the cluster with the specified network interface address.
cl_getclusteridbyifname	Returns the cluster ID of the cluster with the specified network interface name.
cl_getclusters	Returns information about all operational clusters.
cl_isclusteravail	Returns the status of the cluster with the specified cluster ID.

Node Information Requests

The following node information requests return information about the nodes in the cluster:

cl_bestroute	Returns the local/remote IP address pair that indicates the most direct route to the specified node from the local machine.
cl_getlocalid	Returns the cluster ID and node name of the host making the request.
cl_getnode	Returns information about the node specified by a cluster ID/node name pair.
cl_getnodeaddr	Returns the IP address associated with the specified cluster ID/node name pair.
cl_getnodenamebyifaddr	Returns the name of the node with the specified cluster ID/interface address pair.

cl_getnodenamebyifname	Returns the name of the node with the specified cluster ID/interface name pair.
cl_getnodemap	Returns all known information about all the nodes in a specified cluster.
cl_getprimary	Returns the node name of the primary Cluster Manager for the specified cluster.
cl_isnodeavail	Returns the status of the specified node.

Network Interface Information Requests

The following network interface information requests return information about the interfaces connected to a node:

cl_getifaddr	Returns the interface address of the interface with the specified cluster ID and name.
cl_getifname	Returns the interface name of the interface with the specified cluster ID and address.
cl_isaddravail	Returns the status of the specified network interface.

Network Information Requests

The following network information requests return information about networks that are part of a cluster:

cl_getnetmap	Returns all known information about all the networks in the cluster.
cl_getnet	Returns information about a specific network.
cl_getnetbyname	Returns information about a specific, named network.
cl_getnetsbytype	Returns information about any networks configured with the specified type (such as ether, token, hps).
cl_getnetsbyattr	Returns information about any networks configured with the specified attribute (public, private, or non-IP (serial)).
cl_getnetstatebynode	Returns the state of the specified network on the specified node.

Event Notification Requests

The following event notification routines return information about cluster, node, or network events:

cl_getevent	Gets information when an event signal is received, and returns an event notification message received from Clinfo.
cl_registereventnotify	Registers a list of event notification requests with Clinfo, and returns a signal to the calling process when a registered event occurs.
cl_unregistereventnotify	Unregisters a list of event notification requests with Clinfo.

Resource Group Information Requests

The following resource group information requests return information about cluster resource groups:

cl_getgroupmap	Returns all known information about all the resource groups in the specified cluster.
cl_getgroup	Returns all information about the specific resource group in the specified cluster.
cl_getgroupsbynode	Returns all the resource groups in which the specified node is a member.
cl_getgroupnodestate	Returns the state of the specified group on the specified node.

Site Information Requests

The following site information requests return information about sites configured in an HACMP cluster:

cl_getsitemap	Returns all known information about all the sites in the cluster.
cl_getsite	Returns information about a specific site.
cl_getsitebyname	Returns information about a specific, named site.
cl_getsitebypriority	Returns information about any sites configured with the specified priority.

Utilities

The Clinfo C API has the following utility routines:

cl_errmsg	Returns the text for a cluster error code for a single-threaded application.
cl_errmsg_r	Returns the text for a cluster error code for a multi-threaded application.
cl_perror	Writes a message to standard error that describes the specified error code.

cl_initialize Routine

The **cl_initialize()** routine was used in prior releases to attach to shared memory. Beginning with HACMP 5.3, **cl_initialize()** no longer has any function and always returns **CLE_OK**. **cl_initialize()** is provided for backward compatibility only. Do **not** use this routine for new programs.

cl_errmsg Routine

Syntax

```
char *cl_errmsg (int status)
```

Description

The **cl_errmsg** routine takes a status code returned by Clinfo and returns the text for that error code.

Parameters

status	A cluster information error status.

Status Codes

A null-terminated error string.

For example, the string:

```
"Invalid status"
```

if the status parameter does not describe a valid cluster error code.

Example

```
  char *msg;
  msg = cl_errmsg(CLE_BADARGS);
  if (strcmp(msg, "Invalid status") != 0) {
      printf("CLE_BADARGS means %s", msg);
  } else {
      printf("Can't lookup CLE_BADARGS");
  }
```

cl_errmsg_r Routine

This is a thread-safe version of the **cl_errmsg** routine. If you have a multi-threaded application, you must use this routine.

Syntax

```
char *cl_errmsg_r (int status, char cbuf)
```

Description

The **cl_errmsg_r** routine takes a status code returned by Clinfo and returns the text for that error code.

Parameters

status	A cluster information error status.
cbuf	Storage for the returned message must be at least **CL_ERRMSG_LEN** long (enough for 128 characters).

Status Codes

A null-terminated error string.

For example, the string:

```
"Invalid status"
```

if the status parameter does not describe a valid cluster error code.

Example

```
char *msg;
char cbuf[CL_ERRMSG_LEN];
msg = cl_errmsg_r(CLE_BADARGS, cbuf);
if (strcmp(msg, "Invalid status") != 0) {
   printf("CLE_BADARGS means %s", msg);
} else {
   printf("Can't lookup CLE_BADARGS");
}
```

cl_perror Routine

Syntax

```
void cl_perror (int status, char *message)
```

Description

The **cl_perror** routine writes a message that describes a specified error code to standard error. **cl_perror** places the supplied error string before the error message, and places a colon following the error message.

For example, specifying:

```
cl_perror(CLE_IVNODENAME, "Can't service this request");
```

yields the output:

```
Can't service this request:Illeagle Node Name.
```

The **cl_perror** routine is useful for generating an error message from the status code returned by a Clinfo request. If a status is provided that is not a valid cluster error code, the **cl_perror** routine writes the string:

```
Error n
```

where `n` is the value of the specified status code.

Parameters

status	A cluster error code.
message	The message that will proceed the status description.

Example

```
struct cl_node nodebuf;
  int clusterid = 99999999;      /* invalid cluster id */
  int status;
  char nodename[CL_MAXNAMELEN];

  if ( (status = cl_getnode (clusterid, nodename, &nodebuf)) < 0 ) {
      cl_perror(status, "can't get node information");
  }
```

cl_alloc_clustermap Routine

Syntax

```
int cl_alloc_clustermap (struct cl_cluster **clustermap)
```

Description

The **cl_alloc_clustermap** routine allocates storage for a list of clusters. This routine must be called before calling the **cl_getclusters** routine.

After calling the **cl_getclusters** routine, when you are done call the **cl_free_clustermap** routine to free the storage.

Parameters

clustermap	The base pointer for the clustermap.

Status Codes

CLE_OK	The request completed successfully.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for the clustermap argument.

Example

See the example for the **cl_getclusters Routine**.

cl_alloc_groupmap Routine

Syntax

```
int cl_alloc_groupmap (struct cl_group **groupmap);
```

Description

The **cl_alloc_groupmap** routine allocates storage for a list of resource group descriptors. This routine must be called before calling the **cl_getgroups** routine.

After calling the **cl_getgroups** routine, free the storage when you are done by calling the **cl_free_groupmap** routine.

Parameters

groupmap	The base pointer for the group map.

Status Codes

CLE_OK	The request completed successfully.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for the groupmap argument.

Example

See the example for the **cl_getgroupmap Routine**.

cl_alloc_netmap Routine

Syntax

```
int cl_alloc_netmap (struct cl_site **netmap)
```

Description

Allocates storage for a series of network information structures.

Parameters

netmap	A pointer to a structure **cl_net** pointer where a pointer to the newly allocated storage is returned.

Status Codes

CLE_OK	The request complete successfully.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for the **netmap** parameter.

Example

See the example for the **cl_getnetmap Routine**.

cl_alloc_nodemap Routine

Syntax

```
int cl_alloc_nodemap (struct cl_node **nodemap)
```

Description

The **cl_alloc_nodemap** routine allocates storage for a list of nodes and the interfaces associated with each node. This routine should be called before calling the **cl_getnodemap** routine.

After calling the **cl_getnodemap** routine, free the storage by calling the **cl_free_nodemap** routine when you are done.

Parameters

nodemap	The base pointer for the nodemap.

Status Codes

CLE_OK	The request completed successfully.
CLE_BADARGS	Missing or invalid parameters.

Example

See the example for the **cl_getnodemap Routine**.

cl_alloc_sitemap Routine

Syntax

```
int cl_alloc_sitemap (struct cl_net **sitemap)
```

Description

Allocates storage for a series of site information structures.

Parameters

sitemap	A pointer to a structure **cl_site** pointer where a pointer to the newly allocated storage is returned.

Status Codes

CLE_OK	The request complete successfully.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for the **sitemap** parameter.

Example

See the example for the **cl_getsitemap Routine**.

cl_bestroute Routine

Syntax

```
int cl_bestroute (int clusterid, char *nodename,
   struct sockaddr_in *laddr, struct sockaddr_in *raddr)
```

Description

The **cl_bestroute** routine returns the local/remote IP address pair for the most direct route to the specified node. This routine relies on the netmask value to match network interface pairs.

The route returned by the **cl_bestroute** routine depends on the host making the request. Clinfo first builds a list of all working network interfaces on the local node, and then compares this list to the available interfaces on the specified node. The routine sorts interfaces by network and attempts to match the first working interface on the network with the first working interface on the network on the remote node. If private networks are defined, these networks are considered before non-private networks are considered.

If a pair of local and remote interfaces exist that are on the same network, they are returned in the **laddr** and **raddr** parameters. Otherwise, an interface on the specified node is chosen as the remote interface, and the first local interface found is returned as the local end of the route.

Parameters

clusterid	The cluster ID of the desired node.
nodename	The name of the desired node.
laddr	Storage for the returned local network interface address.
raddr	Storage for the returned remote network interface address.

Status Codes

CLE_OK	The request completed successfully.
CLE_BADARGS	Missing or invalid parameters.
CLE_NOROUTE	No route is available.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.

Example

```
int clusterid = 1113325332;
  int status;
  char nodename[CL_MAXNAMELEN] = "node1";
  struct sockaddr_in laddr, raddr;

  status = cl_bestroute(clusterid, nodename, &laddr, &raddr);
  if (status != CLE_OK) {
      cl_perror(status, "can't get route");
  } else {
      printf("best route to node %s is from ", nodename);
      printf("%s to ", inet_ntoa(laddr.sin_addr));
      printf("%s\n", inet_ntoa(raddr.sin_addr));
  }
```

cl_free_clustermap Routine

Syntax

```
void cl_free_clustermap (struct cl_cluster *clustermap)
```

Description

The **cl_free_clustermap** routine frees the storage previously allocated for the list of clusters by calling **cl_alloc_clustermap**.

Parameters

clustermap	A pointer to the clustermap to be freed.

Example

See the example for the **cl_getclusters Routine**.

cl_free_groupmap Routine

Syntax

```
void cl_free_groupmap (struct cl_group *groupmap);
```

Description

The **cl_free_groupmap** routine frees the storage previously allocated by a call to **cl_alloc_groupmap**.

Parameters

groupmap	A pointer to the group map to be freed.

Example

See the example for the **cl_getgroupmap Routine**.

cl_free_netmap Routine

Syntax

```
void cl_free_netmap (struct cl_net *netmap)
```

Description

Frees the storage previously allocated by a call to **cl_alloc_netmap**.

Parameters

netmap	A pointer to the netmap to be freed.

Status Codes

None.

Example

See the example for the **cl_getnetmap Routine**.

cl_free_nodemap Routine

Syntax

```
int cl_free_nodemap (struct cl_node *nodemap)
```

Description

The **cl_free_nodemap** routine frees the storage previously allocated by calling the **cl_alloc_nodemap** routine.

Parameters

nodemap	A pointer to the nodemap to be freed.

Example

See the example for the **cl_getnodemap Routine**.

cl_free_sitemap Routine

Syntax

```
void cl_free_sitemap (struct cl_site *sitemap)
```

Description

Frees the storage previously allocated by a call to **cl_alloc_sitemap**.

Parameters

sitemap	A pointer to the sitemap to be freed.

Status Codes

None.

Example

See the example for the **cl_getsitemap Routine**.

cl_getcluster Routine

Syntax

```
int cl_getcluster (int clusterid, struct cl_cluster *clusterbuf)
```

Description

The **cl_getcluster** returns information about the cluster specified by the cluster ID.

Parameters

clusterid	The cluster ID of the desired cluster.
clusterbuf	Storage for the returned cluster information.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

```
int clusterid = 1113325332;
int status;
struct cl_cluster cluster;

status = cl_getcluster(clusterid, &cluster);
if (status != CLE_OK) {
   cl_perror(status, "Can't get cluster information");
} else {
        printf("cluster id: %d\n",cluster.clc_clusterid);
        printf("cluster name: %s\n",cluster.clc_name);
        printf("state= %d [%s]\n",
          cluster.clc_state,
          get_state(cluster.clc_state));
        printf("substate= %d\n",cluster.clc_substate);
        printf("primary= %s\n",cluster.clc_primary);
        printf("nodes= %d, sites= %d, groups= %d, networks= %d\n",
          cluster.clc_number_of_nodes,
          cluster.clc_number_of_sites,
          cluster.clc_number_of_groups,
          cluster.clc_number_of_networks);
}
```

cl_getclusterid Routine

Syntax

```
int cl_getclusterid (char *clustername)
```

Description

The **cl_getclusterid** routine returns the cluster ID of the cluster with the specified name.

Parameter

clustername	The name of the desired cluster.

Status Codes

A non-negative number, which represents the cluster ID, signifies success. Otherwise, one of the following error status codes:

CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVCLUSTERNAME	The request specified an invalid cluster name.

Example

```
int clusterid = 1113325332;
  char clustername[CL_MAXNAMELEN] = "site1";

  clusterid = cl_getclusterid (clustername);
  if (clusterid < 0) {
      cl_perror (clusterid, "can't get cluster ID");
  } else {
      printf ("cluster %s has id %d\n", clustername, clusterid);
  }
```

cl_getclusteridbyifaddr Routine

Syntax

```
int cl_getclusteridbyifaddr (struct sockaddr_in *addr)
```

Description

Returns the cluster ID of the cluster with the specified network interface address.

Parameters

addr	The network interface address whose cluster ID is desired.

Status Codes

A non-negative number, which represents the cluster ID, signifies success. Otherwise, one of the following error status codes:

CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVADDRESS	The request specified an invalid network interface address.

Example

```
char ifaddr[CL_MAXNAMELEN] = "9.57.28.23";
  int clusterid;
  struct sockaddr_in addr;

  /*
  * inet_addr converts addrs to
  * Internet numbers.
  */

  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = inet_addr (ifaddr);
  clusterid = cl_getclusteridbyifaddr (&addr);
  if (clusterid < 0) {
      cl_perror (clusterid,"can't get cluster ID");
  } else {
      printf("cluster id w/ interface address %s is %d\n",
      inet_ntoa (addr.sin_addr.s_addr), clusterid);
  }
```

cl_getclusteridbyifname Routine

Syntax

```
int cl_getclusteridbyifname (char *interfacename)
```

Description

Returns the cluster ID of the cluster with the specified network interface name.

Parameters

interfacename	The network interface name whose cluster ID is desired.

Status Codes

A non-negative number, which represents the cluster ID, signifies success. Otherwise, one of the following error status codes:

CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

```
int clusterid;
char interfacename[CL_MAXNAMELEN] = "geotest9";

clusterid = cl_getclusteridbyifname (interfacename);
if (clusterid < 0) {
    cl_perror (clusterid, "can't get cluster id");
} else {
    printf ("cluster id w/ interface named %s is %d\n",
    interfacename, clusterid);
}
```

cl_getclusters Routine

Syntax

```
int cl_getclusters (struct cl_cluster *clustermap)
```

Description

Returns information about all working clusters.

Parameters

clustermap	Returned cluster information. You should allocate storage for this information by calling **cl_alloc_clustermap** before calling this routine. Then free this storage space by calling **cl_free_clustermap** when you are done.

Status Codes

The routine returns the number of active clusters. If the routine is unsuccessful, it returns one of the following error status codes:

CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.

Example

This example uses the **cl_errmsg** routine to illustrate proper programming for a single-threaded application. If your program is multi-threaded, you must use the **cl_errmsg_r** routine.

```
  int i;
  int numClusters = -1;
  char cbuf[CL_ERRMSG_LEN];
  struct cl_cluster *clustermap;

  cl_alloc_clustermap (&clustermap);
  numClusters = cl_getclusters(clustermap);
  if(numClusters < 0)
  {
      printf("cl_getclusters: (failed) %s\n",
      cl_errmsg(numClusters));
/*
** for threadsafe compilation use:

      cl_errmsg_r(numClusters,cbuf));
*/
  }
  else
  {
      printf("there are currently %d running clusters\n", numClusters);
      for(i=0; i < numClusters; i++)
       {
          printf("\n  cluster id: %d\n",clustermap[i].clc_clusterid);
          printf("  cluster name: %s\n",clustermap[i].clc_name);
          printf("  state= %s\n",get_state(clustermap[i].clc_state));
          printf("  substate= %d\n",clustermap[i].clc_substate);
          printf("  primary= %s\n",clustermap[i].clc_primary);
          printf("  nodes= %d, sites= %d, groups= %d, networks= %d\n",
            clustermap[i].clc_number_of_nodes,
            clustermap[i].clc_number_of_sites,
            clustermap[i].clc_number_of_groups,
            clustermap[i].clc_number_of_networks);
       }
  }
  cl_free_clustermap(clustermap);
```

cl_getevent Routine

Syntax

```
int cl_getevent (struct cli_en_msg_t * en_msg)
```

Description

The **cl_getevent** routine returns an event notification message. The caller should issue this request only after a signal, as specified in a previous **cl_registereventnotify** request, is received.

Parameters

en_msg	An event notification message buffer, large enough to hold one event notification message. Upon successful return, this buffer contains the received event, cluster, and, if applicable, the network and node identifications.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.

Example

See the example for the **cl_registereventnotify Routine**.

cl_getgroup Routine

Syntax

```
int cl_getgroup (int clusterid, char *groupname,
       struct cl_group *groupbufp);
```

Description

The **cl_getgroup** routine returns information about the specified resource group in the specified cluster.

Parameters

clusterid	The cluster ID of the desired cluster.
groupname	The name of the resource group.
groupbufp	A pointer to a **cl_group** structure where the information is returned.

Status Codes

CLE_OK	Success.
CLE_BADARGS	Missing or invalid argument(s).
CLE_SYSERR	System error.
CLE_NOCLINFO	No cluster information available.
CLE_IVCLUSTERID	Invalid cluster ID.
CLE_IVNODENAME	Invalid resource group name.

Example

```
  int clusterid = 1113325332;
  int status, j;
  char* groupname = "rg01";
  struct cl_group group;

  status = cl_getgroup(clusterid, groupname, &group);
  if (status != CLE_OK){
      cl_perror(status, "can't get resource group information");
  } else {
      printf("resource group %s has %d nodes.\n",
      group.clg_name,
      group.clg_num_nodes);
      for(j=0; j < group.clg_num_nodes; j++){
            printf("node w/ id %d is in state %d [%s]\n",
            group.clg_node_ids[j],
            group.clg_node_states[j],
/* user defined function char* cvrt_rg_state(enum cl_resource_states
state)
** to convert state id numbers to text
*/
           cvrt_rg_state(group.clg_node_states[j]));
      }
  }
```

cl_getgroupmap Routine

Syntax

```
int cl_getgroupmap (int clusterid, struct cl_group *groupmap)
```

Description

The **cl_getgroupmap** routine returns information about the resource groups in a cluster. You should call the **cl_alloc_groupmap** before calling this routine to reserve the storage in memory. You should call **cl_free_groupmap** after calling this routine.

Parameters

clusterid	The cluster ID of the desired cluster.
groupmap	Storage for the returned resource group information.

Status Codes

The request completed successfully if it returns a non-negative number (number of groups in the cluster).

CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

```
int i,j;
int clusterid = 1113325332;
int groups;
char cbuf[CL_ERRMSG_LEN];
struct cl_group *groupmap;

cl_alloc_groupmap(&groupmap);
if (groupmap==NULL){
    printf("unable to allocate storage: cl_alloc_groupmap = NULL\n");
    exit(-1);
}
groups = cl_getgroupmap(clusterid, groupmap);
if(groups < 0) {
    cl_perror(groups, "can't get resource group map");
} else {
    printf("cluster %d has %d resource groups\n", clusterid, groups);
    for(i=0; i < groups; i++){
          printf("resource group %d [%s] has %d nodes\n",
          groupmap[i].clg_group_id,
          groupmap[i].clg_name,
          groupmap[i].clg_num_nodes);
          printf("policies:\n");
          printf("\tstartup  %d \n",groupmap[i].clg_startup_policy);
          printf("\tfallover %d \n",groupmap[i].clg_fallover_policy);
```

```
                printf("\tfallback %d \n",groupmap[i].clg_fallback_policy);
                printf("\tsite     %d \n",groupmap[i].clg_site_policy);
                printf("\tuser     %d \n",groupmap[i].clg_user_policy_name);
                printf("node states:\n");
                for(j=0; j < groupmap[i].clg_num_nodes; j++){
                      printf("\tnode %d is in state %d \n",
                          groupmap[i].clg_node_ids[j],
                          groupmap[i].clg_node_states[j]);
                   }
                printf("resources %d\n",groupmap[i].clg_num_resources);
                for(j=0; j < groupmap[i].clg_num_resources; j++){
                      printf("\tresource %d is in state %d\n",
                          groupmap[i].clg_resource_id[j],
                          groupmap[i].clg_res_state[j]);
                }
          }
      }
```

cl_getgroupnodestate Routine

Syntax

```
enum cl_resource_states cl_getgroupnodestate (int clusterid,
      char *groupname, int nodeid);
```

Description

The **cl_getgroupnodestate** routine returns the state of the specified group on the specified node.

Parameters

clusterid	The cluster ID of the desired cluster.
groupname	Name of the resource group.
nodeid	The ID of the cluster node.

Status Codes

Returns one of the enumerated values in **cl_resource_states**. This enumeration is described in the earlier section Enumerated Type Containing Resource Group State in this chapter.

Example

```
enum cl_resource_states state;
int clusterid = 1113325332;
int nodeid = 1;
char groupname[CL_MAXNAMELEN] = "rg01";

state = cl_getgroupnodestate(clusterid, groupname, nodeid);
if (state == CL_RGNS_INVALID){
    cl_perror(state, "can't get group node state");
} else {
    printf("node w/ id %d is currently in state %d in group %s\n",
    nodeid, state, groupname);
}
```

cl_getgroupsbynode Routine

Syntax

```
int cl_getgroupsbynode (int clusterid, int nodeid,
      struct cl_group **groupbufp, int *groupcountp);
```

Description

The **cl_getgroupsbynode** routine returns information for all resource groups of which the specified node is a member. Note that this routine allocates storage for the return information—the caller must free this storage.

Parameters

clusterid	The cluster ID of the desired cluster.
nodeid	The ID of the cluster node.
groupbufp	A pointer to a **cl_group** structure where the information is returned.
groupcountp	A pointer to an integer where the number of groups returned is stored.

Status Codes

CLE_OK	Success.
CLE_BADARGS	Missing or invalid argument(s).
CLE_SYSERR	System error.
CLE_NOCLINFO	No cluster information available.
CLE_IVCLUSTERID	Invalid cluster ID.
CLE_IVNODENAME	Invalid node ID.

Example

```
int clusterid = 1113325332;
int nodeid = 1;
int status;
int groupcount;
int j;
struct cl_group *groups;

status = cl_getgroupsbynode(clusterid, nodeid, &groups, &groupcount);
if (status != CLE_OK){
    cl_perror(status,"failed to get resource group information");
} else {
    printf("node %d is a member of %d groups:\n",nodeid, groupcount);
    for(j=0; j < groupcount; j++){
        printf("node %d is in group %s\n",
                nodeid, groups[j].clg_name);}
    if (groupcount) free (groups);
}
```

cl_getifaddr Routine

Syntax

```
int cl_getifaddr (int clusterid, char *interfacename,
   struct sockaddr_in *addr)
```

Description

Returns the address of the interface with the specified cluster ID and interface name.

Parameters

clusterid	The cluster ID of the desired network interface.
interfacename	The name of the desired network interface.
addr	Storage for the returned network interface address.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

```
int clusterid = 1113325332;
int status;
char* interfacename = "geotest9";
struct sockaddr_in addr;

status = cl_getifaddr (clusterid, interfacename, &addr);
if (status != CLE_OK) {
    cl_perror (status, "Can't find interface address");
} else {
    printf ("interface address w/ name %s is %s\n",interfacename,
    inet_ntoa (addr.sin_addr.s_addr));
}
```

cl_getifname Routine

Syntax

```
int cl_getifname (int clusterid, struct sockaddr_in *addr,
      char *interfacename)
```

Description

Returns the name of the interface with the specified cluster ID and IP address.

Parameters

clusterid	The cluster ID of the desired network interface.
addr	The IP address of the desired network interface.
interfacename	Storage for the returned network interface name. The interfacename parameter should be no more than CL_MAXNAMELEN characters long.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVADDRESS	The request specified an invalid network interface address.

Example

```
int clusterid = 1113325332;
int status;
struct sockaddr_in addr;
char interfacename[CL_MAXNAMELEN] = "9.57.28.23";

addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr (interfacename);

status = cl_getifname (clusterid, &addr, interfacename);
if (status != CLE_OK) {
    cl_perror (status, "can't find interface name");
}
else {
    printf ("interface name w/ address %s is %s\n",
          inet_ntoa (addr.sin_addr.s_addr), interfacename);
}
```

cl_getlocalid Routine

Syntax

```
int cl_getlocalid (int *clusterid, char *nodename)
```

Description

Returns the cluster ID and the node name of the node making the request. This request returns an error status code for nodes not currently active in the cluster.

Parameters

clusterid	Storage for the returned cluster ID.
nodename	Storage for the returned node name.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVNODE	Node is not a cluster node.

Example

```
int clusterid, status;
char nodename[CL_MAXNAMELEN] = "node1";

status = cl_getlocalid (&clusterid, nodename);
if (status != CLE_OK) {
    if (status == CLE_IVNODE) {
          printf("this node is not a cluster member\n");
    } else {
          cl_perror(status, "can't get local cluster ID");
    }
} else {
    printf ("this node [%s] is a member of cluster id %d\n",
               nodename,clusterid);
}
```

cl_getnet Routine

Syntax

```
int cl_getnet (int clusterid, int netid, struct cl_net *netbuf)
```

Description

Returns information about a specified network.

Parameters

clusterid	The cluster ID of the desired cluster.
netid	The ID of the network.
netbuf	A pointer to a **cl_net** structure where the information is returned.

Status Codes

CLE_OK	Success.
CLE_BADARGS	Missing or invalid argument(s).
CLE_SYSERR	System error.
CLE_NOCLINFO	No cluster information available.
CLE_IVCLUSTERID	Invalid cluster ID.
CLE_IVNETID	Invalid network ID.

Example

```
int clusterid = 1113325332;
int netid = 1;
int status, j;
struct cl_net netmap;

status = cl_getnet(clusterid, netid, &netmap);
if (status == CLE_OK)
{
      printf("information for cluster network %s (id %d):\n",
               netmap.clnet_name, netmap.clnet_id);
      printf("network is type %s\n", netmap.clnet_type);
      printf("network attribute is %d\n", netmap.clnet_attr);
      printf("there are %d nodes on this network\n",
               netmap.clnet_numnodes);
      for (j=0; j<netmap.clnet_numnodes; j++)
      {
              enum cls_state node_state;
              printf("  node id = %d, state = %d,",
              netmap.clnet_node_ids[j],
              netmap.clnet_node_states[j]);
              cl_getnetstatebynode( clusterid, netmap.clnet_id,
              netmap.clnet_node_ids[j], &node_state);
              printf(" state (cl_getnetstatebynode) = %d\n",
                       node_state);
      }
}
```

cl_getnetbyname Routine

Syntax

```
int cl_getnetbyname (int clusterid, char *netname,
      struct cl_net *netbuf)
```

Description

Returns information about a specific, named network.

Parameters

clusterid	The cluster ID of the desired cluster.
netname	The name of the network.
netbuf	A pointer to a **cl_net** structure where the information is returned.

Status Codes

CLE_OK	Success.
CLE_BADARGS	Missing or invalid argument(s).
CLE_SYSERR	System error.
CLE_NOCLINFO	No cluster information available.
CLE_IVCLUSTERID	Invalid cluster ID.
CLE_IVNETNAME	Invalid network name.

Example

```
char netname[CL_MAXNAMELEN];
int clusterid = 1;
int status;
struct cl_net netmap;

status = cl_getnetbyname(clusterid,netname,&netmap);
if (status != CLE_OK) {
      display_error(status,cmd);
} else {
      printf("network %s is type %s: connected to %d nodes\n",
      netmap.clnet_name,
      netmap.clnet_type,
      netmap.clnet_numnodes);
}
```

cl_getnetmap Routine

Syntax

```
int cl_getnetmap (int clusterid, struct cl_net *netmap)
```

Description

Returns information about all the networks in the cluster.

Parameters

clusterid	The cluster ID of the desired cluster.
netmap	Storage for the returned network information.

Status Codes

The request completed successfully if it returns a non-negative number (number of networks in the cluster).

CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

```
int clusterid = 1113325332;
int num_nets;
int i,j;
struct cl_net *nm;

cl_alloc_netmap(&nm);
num_nets = cl_getnetmap(clusterid, nm);
printf("\n\ndumping netmap with %d nets for cluster %d\n\n",
    num_nets, nm->clnet_clusterid);
for (i=0; i<num_nets; i++)
{
    printf("info for network: %s (%d)\n", nm[i].clnet_name,
             nm[i].clnet_id);
    printf(" type = %s\n", nm[i].clnet_type);
    printf(" attribute = %d\n", nm[i].clnet_attr);
    printf(" state = %d\n", nm[i].clnet_state);
    printf(" number of nodes = %d\n", nm[i].clnet_numnodes);
    for (j=0; j<nm[i].clnet_numnodes; j++)
    {
          printf(" node id = %d , state = %d\n",
          nm[i].clnet_node_ids[j],
          nm[i].clnet_node_states[j]);
     }
}
cl_free_netmap(nm);
```

cl_getnetsbyattr Routine

Syntax

```
int cl_getnetsbyattr (int clusterid, int *netattr,
       struct cl_net **netbuf, int *netcount)
```

Description

Returns information about any networks configured with the specified attribute (public, private or non-IP (serial)).

Parameters

clusterid	The cluster ID of the desired cluster.
netattr	Network attribute for list of networks to be retrieved.
netbuf	A pointer to a **cl_net** structure where the information is returned.
netcount	A pointer to an integer where the number of networks matching the specified type is returned.

Status Codes

CLE_OK	Success. Note that this code may return even if no networks are returned, that is, no networks with the specified attribute exist.
CLE_BADARGS	Missing or invalid argument(s).
CLE_SYSERR	System error.
CLE_NOCLINFO	No cluster information available.
CLE_IVCLUSTERID	Invalid cluster ID.
CLE_IVNETATTR	Invalid network attribute.

Example

```
int status,i;
int netcount;
int clusterid = 1;
struct cl_net *netmap;
enum cl_network_attribute attr;

attr = CL_NET_TYPE_PRIVATE;
status = cl_getnetsbyattr(clusterid, attr, &netmap,&netcount);
if (status != CLE_OK) {
   cl_perror(status,"cl_getnetsbyattr() failed");
} else {
   printf("there are %d private networks in this cluster.\n",netcount);
   for (i=0; i<netcount; i++)
   {
     struct cl_net network;
      status = cl_getnetbyname(clusterid, netmap[i].clnet_name,
                               &network);
      if (status != CLE_OK) {
    cl_perror(status,"cl_getnetbyname() failed");
     } else {
    printf(" network %s is type %s: connected to %d nodes\n",
          network.clnet_name,
          network.clnet_type,
          network.clnet_numnodes);
      }
   }
}
```

cl_getnetsbytype Routine

Syntax

```
int cl_getnetsbytype (int clusterid, char *nettype,
       struct cl_net **netbuf, int *netcount)
```

Description

Returns information about any networks configured with the specified type (such as ether, token, hps).

Parameters

clusterid	The cluster ID of the desired cluster.
nettype	Network type for list of networks to be retrieved. The types of networks are retrieved from the list of Network Interface Modules (NIMs) currently defined to HACMP.
netbuf	A pointer to a **cl_net** structure where the information is returned.
netcount	A pointer to an integer where the number of networks matching the specified type is returned.

Status Codes

CLE_OK	Success. Note that this code may return even if no networks are returned, that is, no networks of the specified type exist.
CLE_BADARGS	Missing or invalid argument(s).
CLE_SYSERR	System error.
CLE_NOCLINFO	No cluster information available.
CLE_IVCLUSTERID	Invalid cluster ID.
CLE_IVNETTYPE	Invalid network type.

Example

```
char net_type[CL_MAXNAMELEN] = "ether";
  int status,i,j;
  int netcount;
  int clusterid = 1113325332;
  struct cl_net *netmap;

  status = cl_getnetsbytype(clusterid, net_type, &netmap, &netcount);
  if (status != CLE_OK) exit(status);
  printf("there are %d networks of type: %s.\n",netcount,net_type);
  for (i=0; i<netcount; i++)
  {
    struct cl_net network;
    status = cl_getnetbyname(clusterid, netmap[i].clnet_name, &network);
    if (status != CLE_OK) exit(status);
    printf("network %s has attribute %d and is connected to %d nodes\n",
    network.clnet_name,
    network.clnet_attr,
    network.clnet_numnodes);
  }
```

cl_getnetstatebynode Routine

Syntax

```
int cl_getnetstatebynode (int clusterid, int netid,
      int nodeid, enum cls_state *state)
```

Description

Returns the state of the specified network on the specified node.

Parameters

clusterid	The cluster ID of the desired cluster.
netid	The network ID for the network state to be retrieved.
nodeid	The node ID for the network state to be retrieved.
netstate	A pointer to an integer where the state of the specified network on the specified node is returned.

Status Codes

CLE_OK	Success.
CLE_BADARGS	Missing or invalid argument(s).
CLE_SYSERR	System error.
CLE_NOCLINFO	No cluster information available.
CLE_IVCLUSTERID	Invalid cluster ID.
CLE_IVNETID	Invalid network ID.
CLE_IVNODEID	Invalid node ID. Note that this return can indicate that the ID is simply invalid (it is not a valid node ID for this cluster) or that the node is not connected (has no interfaces on) the specified network.

Example

See the example for the **cl_getnet Routine**.

cl_getnode Routine

Syntax

```
int cl_getnode (int clusterid, char *nodename,
      struct cl_node *nodebuf);
```

Description

Returns information about the node specified by a cluster ID/node name pair.

Parameters

clusterid	The cluster ID of the desired node.
nodename	The node name of the desired node.
nodebuf	Storage for the returned node information.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.

Example

```
int clusterid = 1113325332;
int status;
char* nodename = "node1";
struct cl_node nodebuf;

status = cl_getnode (clusterid, nodename, &nodebuf);
if (status != CLE_OK) {
    cl_perror(status, "can't get node info");
} else {
    printf("node named %s on cluster %d : id= %d, state= %d\n",
      nodename,
      clusterid,
      nodebuf.cln_nodeid,
      nodebuf.cln_state);
}
cl_node_free(&nodebuf);
```

cl_getnodeaddr Routine

Syntax

```
int cl_getnodeaddr (int clusterid, char *interfacename,
   struct sockaddr_in *addr)
```

Description

Returns the IP address associated with the specified cluster ID/network interface name pair.

Parameters

clusterid	The cluster ID of the desired network interface.
interfacename	The name of the desired network interface.
addr	Storage for the returned network interface address.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

```
int clusterid = 1113325332;
int status;
char* interfacename = "geotest9";
struct sockaddr_in addr;

status = cl_getnodeaddr(clusterid, interfacename, &addr);
if (status != CLE_OK) {
    cl_perror(status, "can't get node addr");
} else {
    printf("address of interface %s on cluster %d is %s\n",
    interfacename, clusterid, inet_ntoa(addr.sin_addr));
}
```

cl_getnodemap Routine

Syntax

```
int cl_getnodemap (int clusterid, struct cl_node *nodemap)
```

Description

The **cl_getnodemap** routine returns information about the nodes in a cluster. You should call **cl_alloc_nodemap** before calling this routine to reserve the storage in memory. You should call **cl_free_nodemap** after calling this routine.

Parameters

clusterid	The cluster ID of the desired cluster.
nodemap	Storage for the returned node information.

Status Codes

The request completed successfully if it returns a non-negative number (number of nodes in the cluster).

CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

```
int clusterid = 1113325332;
int i;
int nodes;
struct cl_node *nodemap;

cl_alloc_nodemap (&nodemap);
if (nodemap==NULL){
    printf("unable to allocate storage: cl_alloc_nodemap = NULL\n");
    exit(-1);
}
nodes = cl_getnodemap(clusterid, nodemap);
if(nodes < 0)
{
    cl_perror(nodes,"can't get node map");
} else {
  printf("cluster %d has %d nodes:\n", clusterid, nodes);
  for(i=0; i < nodes; i++){
    printf("node %s in state %d has %d interfaces\n",
    nodemap[i].cln_nodename,
    nodemap[i].cln_state,
    nodemap[i].cln_nif);
    if(clusterid != nodemap[i].cln_clusterid){
          printf("structure has invalid cluster ID: %d",
          nodemap[i].cln_clusterid);
    }
  }
}
cl_free_nodemap(nodemap);
```

cl_getnodenamebyifaddr Routine

Syntax

```
int cl_getnodenamebyifaddr (int clusterid, struct sockaddr_in *addrp,
      char *nodename)
```

Description

Returns the name of the node with the specified interface address. Clinfo scans the network interfaces on each node in the cluster. If a match is found, the node name for the node associated with that interface address is returned.

Parameters

clusterid	The cluster ID of the desired node.
addr	The network interface address of the desired node.
nodename	The name of the desired node.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVADDRESS	The request specified an invalid network interface address.

Example

```
int clusterid = 1113325332;
int status;
char nodename[CL_MAXNAMELEN];
struct sockaddr_in addr;

addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr ("9.57.28.23");
status = cl_getnodenamebyifaddr (clusterid, &addr, nodename);
if (status !=CLE_OK) {
    cl_perror(status,"can't get node name");
} else {
    printf("node name of interface w/ address %s on
            cluster %d is %s\n",
    inet_ntoa(addr.sin_addr), clusterid, nodename);
}
```

cl_getnodenamebyifname Routine

Syntax

```
int cl_getnodenamebyifname (int clusterid, char *interfacename,
      char *nodename)
```

Description

Returns the node name of the node with the specified interface name. Clinfo scans the network interfaces on each node in the cluster. If a match is found, the node name for that node is returned.

Parameters

clusterid	The cluster ID of the desired node.
interfacename	The network interface name of the desired node.
nodename	The name of the desired node.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

```
int clusterid = 1113325332;
int status;
char nodename[CL_MAXNAMELEN];
char interfacename[CL_MAXNAMELEN] = "geotest9";

status = cl_getnodenamebyifname (clusterid, interfacename, nodename);
if (status != CLE_OK) {
    cl_perror(status,"can't get node name");
} else {
    printf("interface= %s\n",interfacename);
    printf("node name of interface w/ name %s on cluster %d is %s\n",
        interfacename,
        clusterid,
        nodename);
}
```

cl_getprimary Routine

Syntax

```
int cl_getprimary (int clusterid, char *nodename)
```

Description

Returns the node name of the user-designated primary Cluster Manager for the specified cluster.

Parameters

clusterid	The cluster ID whose primary node name is desired.
nodename	The name of the node designated as the primary Cluster Manager node is returned in this argument.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_NOPRIMARY	No information is available about the primary Cluster Manager.
CLE_IVCLUSTER	The cluster is not available.

Example

```
int clusterid = 1113325332;
int status;
char nodename[CL_MAXNAMELEN] = "node1";

status = cl_getprimary (clusterid, nodename);
if (status != CLE_OK) {
    cl_perror(status,"can't get cluster primary");
} else {
    printf ("primary node for cluster %d is %s\n",
    clusterid, nodename);
}
```

cl_getsite Routine

Syntax

```
int cl_getsite(int clusterid, int siteid, struct cl_site *sitebuf)
```

Description

Returns information about a specific site.

Parameters

clusterid	The desired cluster.
siteid	The ID of the cluster.
sitebuf	Holds the returned **cl_site** structure.

Status Codes

CLE_OK	Success.
CLE_BADARGS	Missing or invalid argument(s).
CLE_SYSERR	System error.
CLE_NOCLINFO	No cluster information available.
CLE_IVCLUSTERID	Invalid cluster ID.
CLE_IVSITEID	Invalid site ID

Example

```
int clusterid = 1113325332;
int status;
int siteid = 1;
struct cl_site site;

status = cl_getsite(clusterid, siteid, &site);
if (status == CLE_OK) {
    printf("site %s (%d) has %d nodes and is priority %d\n",
           site.clsite_name,
           site.clsite_id,
           site.clsite_numnodes,
           site.clsite_priority);
} else {
    cl_perror(status,"can't get site information");
}
```

cl_getsitebyname Routine

Syntax

```
int cl_getsitebyname (int clusterid, char *sitename, struct cl_site *sitebuf)
```

Description

Returns information about a specific, named site.

Parameters

clusterid	The desired cluster.
sitename	The name of the site to be returned.
sitebuf	Holds the returned **cl_site** structure.

Status Codes

CLE_OK	Success.
CLE_BADARGS	Missing or invalid argument(s).
CLE_SYSERR	System error.
CLE_NOCLINFO	No cluster information available.
CLE_IVCLUSTERID	Invalid cluster ID.
CLE_IVSITENAME	Invalid site ID

Example

```
int clusterid = 1113325332;
int status;
char sitename[CL_MAXNAMELEN] = "geo9";
struct cl_site site;

status = cl_getsitebyname(clusterid, sitename, &site);
if (status == CLE_OK)
    printf("site %s (%d) has %d nodes and is priority %d\n",
        site.clsite_name,
        site.clsite_id,
        site.clsite_numnodes,
        site.clsite_priority);
```

cl_getsitebypriority Routine

Syntax

```
int cl_getsitebypriority (int clusterid, enum cl_site_priority,
    priority, struct cl_site *sitebuf)
```

Description

Returns information about any sites configured with the specified priority.

Parameters

clusterid	The desired cluster.
priority	One of the enumerated site priority values.
site_buf	Holds the returned site information.

Status Codes

CLE_OK	Success.
CLE_BADARGS	Missing or invalid argument(s).
CLE_SYSERR	System error.
CLE_NOCLINFO	No cluster information available.
CLE_IVCLUSTERID	Invalid cluster ID.
CLE_IVSITEPRIORITY	Invalid site priority.

Example

```
enum cl_site_priority priority = CL_SITE_PRI_PRIMARY;
int clusterid = 1113325332;
int status;
struct cl_site site;

status = cl_getsitebypriority(clusterid, priority, &site);
if (status == CLE_OK)
    printf("site %s (%d) has %d nodes and is priority %d\n",
        site.clsite_name,
        site.clsite_id,
        site.clsite_numnodes,
        site.clsite_priority);
```

cl_getsitemap Routine

Syntax

```
int cl_getsitemap (int clusterid, struct cl_site *sitemap)
```

Description

Returns all known information about all the sites in the cluster.

Parameters

clusterid	The desired cluster.
sitemap	Storage must be allocated prior to the **cl_getsitemap** call using **cl_alloc_sitemap(&sitemap)**. Storage must be freed using **cl_free_sitemap(sitemap)**.

Status Codes

The request completed successfully if it returns a non-negative number (number of sites in the cluster).

CLE_BADARGS	Missing or invalid argument(s).
CLE_SYSERR	System error.
CLE_NOCLINFO	No cluster information available.
CLE_IVCLUSTERID	Invalid cluster ID.

Example

```
int clusterid = 1113325332;
int status;
int i,j;
int nbr_sites;
struct cl_site *sitemap;

cl_alloc_sitemap(&sitemap);
nbr_sites = cl_getsitemap(clusterid, sitemap);
printf("dumping sitemap with %d sites for cluster %d\n\n",
    nbr_sites, sitemap->clsite_clusterid);
for (i=0; i<nbr_sites; i++)
{
    printf("info for site %s (%d)\n",
    sitemap[i].clsite_name, sitemap[i].clsite_id);
    printf(" priority = %d \n", sitemap[i].clsite_priority);
    printf(" backup = %d \n", sitemap[i].clsite_backup);
    printf(" state = %d \n", sitemap[i].clsite_state);
    printf(" number of nodes = %d\n", sitemap[i].clsite_numnodes);
    for (j=0; j<sitemap[i].clsite_numnodes; j++)
    {
          printf(" node id = %d\n",
          sitemap[i].clsite_nodeids[j]);
    }
}
cl_free_sitemap(sitemap);
```

cl_isaddravail Routine

Syntax

```
int cl_isaddravail (int clusterid, char *nodename,
   struct sockaddr_in *addr)
```

Description

Returns the status of the specified network interface.

Parameters

clusterid	The cluster ID of the desired network interface.
nodename	The node name of the desired network interface.
addr	The address of the desired network interface.

Status Codes

CLE_OK	The specified address is available from the given node.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.
CLE_IVADDRESS	The request specified an invalid interface address.
CLE_IVNETIF	The network interface is not available.

Example

```
int clusterid = 1113325332;
int status;
char ifaddr[CL_MAXNAMELEN] = "9.57.28.23";
char nodename[CL_MAXNAMELEN] = "node1";
struct sockaddr_in addr;

addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr (ifaddr);
status = cl_isaddravail (clusterid, nodename, &addr);
if (status != CLE_OK) {
    cl_perror(status,"interface not available");
} else {
    printf ("interface address for %s is available\n",
        inet_ntoa (addr.sin_addr.s_addr));
}
```

cl_isclusteravail Routine

Syntax

```
int cl_isclusteravail (int clusterid)
```

Description

Returns status of the cluster with the specified cluster ID.

Parameters

clusterid	The cluster ID of the desired cluster.

Status Codes

CLE_OK	The cluster is available.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_IVCLUSTER	The specified cluster is not available.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

```
int clusterid = 1113325332;
int status;

status = cl_isclusteravail (clusterid);
if (status != CLE_OK) {
    cl_perror (status, "cluster is not available");
} else {
    printf ("cluster %d is available\n", clusterid);
}
```

cl_isnodeavail Routine

Syntax

```
int cl_isnodeavail (int clusterid, char *nodename)
```

Description

Indicates whether the specified node is alive.

Parameters

clusterid	The cluster ID of the desired node.
nodename	The node name of the desired node.

Status Codes

CLE_OK	The node is available from the specified cluster.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.
CLE_IVNODE	The node is not available.

Example

```
int clusterid = 1113325332;
int status;
char nodename[CL_MAXNAMELEN] = "node1";

status = cl_isnodeavail (clusterid, nodename);
if (status != CLE_OK) {
    cl_perror(status,"node is unavailable");
} else {
    printf ("node %s on cluster %d is available\n",
   nodename, clusterid);
}
```

cl_model_release Routine

The **cl_model_release()** routine was used in prior releases to detach shared memory. Beginning with HACMP 5.3, **cl_model_release()** no longer has any function and always returns **CLE_OK**. **cl_model_release()** is provided for backward compatibility only. Do **not** use this routine for new programs.

cl_node_free

Syntax

```
int cl_node_free
```

Description

Deletes the storage previously allocated for the network interfaces associated with the node returned by a previous call to the **cl_getnode** routine. The member **cln_if** will be set to null:

Parameters

nodebifp	node buffer

Status Codes

CLE_OK	The request completed successfully.

Sample Source

```
int clusterid;
struct cl_node nodebuf;
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr("9.57.28.8");

clusterid = cl_getclusteridbyifaddr (&addr);

cl_getnode(clusterid, "ppstest5", &nodebuf);
printf ("node id: %d has %d interfaces \n",
               nodebuf.cln_nodeid, nodebuf.cln_nif);

/* call cl_node_free to release storage for the network */
/* interfaces (part of the cl_node struct) which was    */
/* allocated by cl_getnode */
cl_node_free(&nodebuf);

printf ("after cl_node_free, node id: %d has %d interfaces \n",
    nodebuf.cln_nodeid, nodebuf.cln_nif);
```

Sample Output

```
node id: 1 has 5 interfaces
after cl_node_free, node id: 1 has 0 interfaces
```

cl_registereventnotify Routine

Syntax

```
#include <signal.h>
int cl_registereventnotify (int num_reqs, struct cli_enr_req_t*enr_list)
```

Description

The **cl_registereventnotify** routine registers a list of event notification requests with Clinfo. Each request specifies an event ID, a cluster ID, a signal ID, and if applicable, a node name and/ or a network ID. If the routine is successful, Clinfo signals the calling process when an event occurs, which fits the specified description. When this signal is received, the **cl_getevent** routine may be called to get information about the event that just occurred.

You can register to receive notification of all network and cluster events by specifying -1 as the event ID. If you register to receive notification of all such events, you cannot unregister notification of specific events; you must unregister all.

Note that you cannot use the -1 event ID for events that include node name identification. Specify a NULL string to register for all node names. Currently you cannot specify an individual network ID. Use -1 as the event ID.

This routine does not work with a cluster ID of 0. Assign another number as the cluster ID.

Parameters

num_reqs	The number of event notification registration requests being made (limit is 10).
enr_list	The list of event notification registration requests. The events that may be requested include the following: **CL_SWAPPING_ADAPTER**. The specified adapters are being swapped. This event requires cluster, node, and network identification. A value of -1 for cluster and network identifications indicates all such components. Specify a NULL string for all node names. This corresponds to the **swap_adapter** event in **hacmp.out**. **CL_SWAPPED_ADAPTER**. The specified adapters have been swapped. This event requires cluster, node, and network identification. A value of -1 for cluster and network identifications indicates all such components. Specify a NULL string for all node names. This corresponds to the **swap_adapter_complete** event in **hacmp.out**.

CL_JOINING_NETWORK. The specified network is in the process of joining. This event requires cluster and network identification. A value of -1 for either of these identifications indicates all such components. This corresponds to the **network_up** event in **hacmp.out**.

CL_JOINED_NETWORK. The specified network is up. This event requires cluster and network identification. A value of -1 for either of these identifications indicates all such components. This corresponds to the **network_up_complete** event in **hacmp.out**.

CL_FAILING_NETWORK. The specified network is going down. This event requires cluster and network identification. A value of -1 for either of these identifications indicates all such components. This corresponds to the **network_down** event in **hacmp.out**.

CL_FAILED_NETWORK. The specified network is down. This event requires cluster and network identification. A value of -1 for either of these identifications indicates all such components. This corresponds to the **network_down_complete** event in **hacmp.out**.

CL_JOINING_NODE. The specified node is coming up. This event requires cluster and node identification. This corresponds to the **node_up** event in **hacmp.out**.

CL_JOINED_NODE. The specified node is up. This event requires cluster and node identification. This corresponds to the **node_up_complete** event in **hacmp.out**.

CL_FAILING_NODE. The specified node is going down. This event requires cluster and node identification. This corresponds to the **node_down** event in **hacmp.out**.

CL_FAILED_NODE. The specified node is down. This event requires cluster and node identification. This corresponds to the **node_down_complete** event in **hacmp.out**.

CL_NEW_PRIMARY. The specified cluster has a new primary node. This event requires the cluster ID. Note that the concept of a primary node is an attribute from prior releases and may not correspond to the role of the node with respect to the application.

CL_STABLE. The specified cluster has stabilized. This event requires cluster identification. A value of -1 indicates all such components.

	CL_UNSTABLE. The specified cluster has destabilized. This event requires cluster identification. A value of -1 indicates all such components.
SIGNALS	You may specify any appropriate UNIX signal; familiarity with signals is assumed. SIGUSR1 and SIGUSR2 are suggested.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.
CLE_IVREQNUM	An invalid number of event notification registration requests was specified (more than 10).
CLE_IVNETID	An invalid network identification was specified.
CLE_IVEVENTID	An invalid event identification was specified.

Example

The following example illustrates the use of the **cl_registereventnotify, cl_unregistereventnotify**, and **cl_getevent** routines in a simple test program.

In this example, the application registers to be notified by a SIGUSR1 signal when any network connected to node2 of cluster 83 experiences problems serious enough to cause a FAILING_NETWORK event. After registering, the application waits for the event. When the event occurs, Clinfo sends the SIGUSR1 signal to the application; the application then sends for the information about the event using **cl_getevent**, and prints out the information received. Finally, the application unregisters for notification of this event.

```
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
#include <cluster/clinfo.h>
volatile int no_signal = 1;

/* Signal handler for catching event notification signal */
void
catch_sig(int sig)
{
   no_signal = 0;
}
```

```
int
main(int argc, char *argv[])
{
       int ret_code, i;
       struct cli_enr_req_t en_req;  /*  Event notification request  */
       struct cli_en_msg_t  en_msg;  /*  Event notification message */
       en_req.event_id = CL_FAILING_NETWORK;/* specify a failing
               network  event  */
       en_req.cluster_id = 83;
       strcpy (en_req.node_name,"node2");
       en_req.net_id = 1;       /*  no net id necessary
               for this event  */
       en_req.signal_id = SIGUSR1;/*  Request to register for event
               notification */

       if(ret_code = cl_registereventnotify((int) 1, &en_req)!=CLE_OK )
       {
               printf("cl_en_test: cl_registereventnotify failed
               with error %d.",ret_code);
       exit(1);
       }
/* Set up a signal handler to catch the event notification
   signal from Clinfo   */

       ret_code = signal(SIGUSR1, catch_sig);
       if ( ret_code < 0 ){
               perror("cl_en_test");
               exit(1);
       }
    /*  Wait for signal  */

       printf("cl_en_test: waiting for signal from Clinfo.");
       while (no_signal){
       pause();}
    /*  Execution will start here after catch_sig executes when
the signal is received.  Get the event notification message.  */

       if ( ret_code = cl_getevent(&en_msg) != CLE_OK )
       {
       printf("cl_en_test: cl_getevent failed with error %d.",
                ret_code);
       exit(1);
       }
/* Print out the event notification information received */

       printf("cl_en_test: Event notification message received
               from Clinfo:");
       printf("cl_en_test: Event id = %d", en_msg.event_id);
       printf("cl_en_test: Cluster id = %d", en_msg.cluster_id);
       printf("cl_en_test: Node name = %s", en_msg.node_name);
       printf("cl_en_test: Net id = %d", en_msg.net_id);
/*  Request to unregister the event notification  */
    if ( (ret_code = cl_unregistereventnotify((int) 1,&en_req))!=CLE_OK)
       {
       printf("cl_en_test: cl_unregistereventnotify failed with
       error %d.", ret_code);
       exit(1);
       }
}
```

cl_unregistereventnotify Routine

Syntax

```
int cl_unregistereventnotify (int num_reqs,
      struct cli_enr_req_t *enr_list)
```

Description

The **cl_unregistereventnotify** routine unregisters a list of event notification requests with Clinfo. Each request specifies an event identification, a cluster identification, a signal identification, and if applicable, a node and/or network identification. If the routine is successful, Clinfo deletes registration of all events that fit the specified description.

If you registered to receive notification of all network or cluster events by specifying -1 as the event ID, you must unregister all. You cannot register for all, then unregister individual events. If you registered for specific events, then unregister for those events individually.

Parameters

See parameters for the **cl_registereventnotify Routine**.

Status Codes

See status codes for the **cl_registereventnotify Routine**.

Example

See the example for the **cl_registereventnotify Routine**.

Chapter 3: Clinfo C++ API

The Clinfo C++ API is an object-oriented interface that you can use in a C++ application to get status information about an HACMP for AIX cluster. This chapter describes the specific C++ language objects and methods available in the Clinfo C++ API.

In addition to this chapter read Chapter 2: Clinfo C API, which describes the C API. The C++ API routines call the C API routines.

See the *IBM C for AIX C/C++ Language Reference* for more information on C++ and the AIX XL C++ compiler.

Note: If you upgrade to HACMP 5.3 and will be using the new version of Clinfo, you do not need to recompile because HACMP 5.3 contains changes to Clinfo that incorporates version information.

Note: HACMP 5.3 eliminated **cl_registerwithclsmuxpd()** API. Any pre or post-script compiled with this API will fail to load. Use application monitoring instead of the **cl_registerwithclsmuxpd()** routine. Refer to the section on Application Monitoring in Chapter 2: Initial Cluster Planning in the *Planning Guide*.

Clinfo C++ Object Classes

The Clinfo C++ API has the following object classes: clusters, nodes, network interfaces, and resource groups. Each network interface belongs to a single node; each node and each resource group belongs to a single cluster; this dictates the class structure. The cluster class is the most general, and the interface class is the most specific.

The grouping of functions into classes depends on the data that the functions use, with functions being placed into the most general possible class. Note that there are functions named **CL_getclusterid** in both the network interface and cluster classes. These are separate functions with the same name but distinguished by class. This naming convention is standard in C++.

All values are passed to the caller using the normal function return values, rather than function parameters passed by reference. When these descriptions talk about data being passed in, it is implicit that this data is coming from the object of which the function is a member.

Status is always returned in a **CL_status** arg, which is passed by reference and should be checked for error conditions. Return values are always data. An exception to this is the **CL_isavail()** function, whose primary return data is status, so it returns status rather than using the **CL_status** arg.

The Clinfo C++ API includes two functions that do not belong to any of the classes stated above. **CL_getallinfo** operates on an array of clusters rather than a cluster object. It returns an array of clusters, and the array pointer is passed by reference. **CL_getlocalid** operates on the local node rather than on a node object. The local host returns it’s own name.

Using the Clinfo C++ API in an Application

This section describes how to use the Clinfo C++ API in an application.

HACMP for AIX includes separate libraries for multi-threaded and for single-threaded applications. Be sure to link with the appropriate library for your application.

Linking to the Clinfo C API

It is possible for C++ programs to call the Clinfo C API by using appropriate linkage directives, for example:

```
extern "C" int cl_getclusterid (char *);
```

See the *AIX XL C++/6000 V1.1.1 Language Reference* for more information.

Linkage directives that allow C++ programs to call the Clinfo C API are included in **clinfo.h**. However, if you want a C++ API that is more object oriented, you may use the Clinfo C++ API.

The Clinfo C++ API defines object classes for cluster, node, and network interface. All Clinfo C++ functions that retrieve data from these objects are member functions (sometimes called methods) of these classes.

Header Files

You must specify the following include directive in each source module that uses the Clinfo C++ API:

```
#include <cluster/clinfo.H>
```

The libclpp.a and libclpp_r.a Libraries

You must add the following directives to the object load command of a *single-threaded* application that uses the Clinfo C++ API:

```
-lclpp -lcl -lclstr
```

You must add the following directive to the object load command of a *multi-threaded* application that uses the Clinfo C API:

```
-lclpp_r -lcl_r  -lclstr_r
```

The **libclpp.a** and **libclpp_r.a** libraries hold the routines that support the Clinfo C++ API; the **libcl.a** and **libcl_r.a** libraries contain the routines that support the Clinfo C API.

To compile a C++ program under AIX, use the xlC compiler.

Note: Clinfo's **cluster.es.client.lib** library now contains the **libcl.a** with both 32- and 64-bit objects. You must recompile/relink your application in a 64-bit environment to get a 64-bit application using the Clinfo API.

Constants

The Clinfo C++ API routines use the following constants, defined in the **clinfo.h** file:

CL_MAXNAMELEN	The maximum length of a character string naming a cluster, node, or interface (256). The value of **CL_MAXNAMELEN** is the same as **MAXHOSTNAMELEN**, defined in the **sys/param.h** file.
CL_MAX_EN_REQS	The maximum number of event notification requests allowed (10).
CL_ERRMSG_LEN	Maximum error message length (128 characters).
CLSMUXPD_SVC_PORT	The constants **CL_MAXCLUSTERS**, **CL_MAXNODES**, and **CL_MAXNETIFS** used in previous versions of the software are replaced by macros that provide the current sizes of the various arrays in the data structure. The macros have the same name as the constants they replace. These macros cannot be used as array bounds in declaration statements, as the constants were used.
CL_MAXCLUSTERS	Provides current size of space in shared memory for the array holding information about the number of clusters.
CL_MAXNODES	Provides current size of space in shared memory for the array holding information about the number of nodes in a cluster.
CL_MAXNETIFS	Provides current size of space in shared memory for the array holding information about the number of interfaces attached to a node.
CL_MAXGROUPS	Provides current size of space in shared memory for the array holding information about the number of resource groups.

As an example, the following code fragment illustrates use of the constant **CL_MAXNODES** to size an array of cluster objects. An example of how to replace this with a call to the routine of the same name follows.

```
CL_cluster clusters[CL_MAXNODES];
CL_cluster *ret = &clusters[0];
ret = CL_getallinfo(ret, s);
if (s < 0)
   cl_errmsg(s);
for (int i=0; i<CL_MAXNODES; i++) {
   printf("[%d] cl %d", i, ret[i].clc_clusterid);
   printf(" st %d", ret[i].clc_state);
   printf(" su %d", ret[i].clc_substate);
   printf(" pr %d", ret[i].clc_primary);
   printf(" na %s", ret[i].clc_name.name);
}
```

You no longer use the constant **CL_MAXNODES**.

```
CL_cluster clusters[8];
CL_cluster *ret = &clusters[0];
int numclus;
   numclus = CL_getallinfo(ret, s);
if (s < 0)
   cl_perror(s, progname);
printf("number of clusters found: %d", numclus);
for (int i=0; i<numclus; i++) {
   printf("[%d] cl %d", i, ret[i].clc_clusterid);
   printf(" st %d", ret[i].clc_state);
   printf(" su %d", ret[i].clc_substate);
   printf(" pr %s", ret[i].clc_primary);
   printf(" na %s", ret[i].clc_name.name);
}
```

Data Types and Structures

The Clinfo C++ API uses the following data types and structures, defined in the **clinfo.H** file. The **clinfo.H** file also includes the **sys/types.h**, **netinet/in.h**, and **clinfo.h** files.

Basic Data Types and Class Definitions

The following data types translate the C data types defined in the **clinfo.h** file to C++.

```
typedef int CL_clusterid;
typedef int CL_nodeid;
typedef int CL_ifid;
typedef struct sockaddr_in CL_ifaddr;
typedef enum cls_state CL_state;
typedef enum clss_substate CL_substate;
typedef int CL_status;
typedef int CL_groupid;
typedef enum cl_rg_policies CL_rg_policies;
typedef enum cl_resource_states CL_resource_states;
class CL_clustername {public: char name[CL_MAXNAMELEN]; };
class CL_nodename {public: char name[CL_MAXNAMELEN]; };
class CL_ifname {public: char name[CL_MAXNAMELEN]; };
class CL_route {
   public:
   CL_ifaddr localaddr;
   CL_ifaddr remoteaddr;
};
class CL_groupname {public: char name[CL_MAXNAMELEN]; };
class CL_user_policy_name {public: char name[CL_MAXNAMELEN]; };
```

Cluster Object Class

Cluster class data and member functions:

```
class CL_cluster  {
   public:
   CL_clusterid clc_clusterid;    // Cluster Id
   CL_state clc_state;            // Cluster State
   CL_substate clc_substate;      // Cluster Substate
   CL_nodename clc_primary;       // Cluster Primary Node
   CL_clustername clc_name;       // Cluster Name
   CL_node *clc_node;             // Pointer to child node array
   CL_group *clc_group;           // pointer to child resource group
   array
   int CL_getallinfo(CL_node*, CL_status&);
   int CL_getgroupinfo(CL_group*, CL_status&);
   CL_clusterid CL_getclusterid(CL_status&);
   CL_cluster CL_getinfo(CL_status&);
   CL_status CL_getprimary(CL_status&, CL_nodename);
   CL_status CL_isavail();
   CL_cluster& operator=(const struct cl_cluster&);
};
```

Network Interface Object Class

Network interface class data and member functions:

```
class CL_netif  {
   public:
   CL_clusterid cli_clusterid;   // Cluster Id
   CL_nodeid cli_nodeid;         // Cluster Node Id (internal)
   CL_nodename cli_nodename;     // Cluster Node name
   CL_ifid cli_interfaceid;      // Cluster Node Interface Id
   CL_state cli_state;           // Cluster Node Interface
                                 // State
   CL_ifname cli_name;           // Cluster Node Interface
                                 // Name
   CL_ifaddr cli_addr;           // Cluster Node Interface IP
                                 // Address
   CL_node *cli_pnode;           // pointer to parent Node
                                 // object
   CL_clusterid CL_getclusterid(CL_status&);
   CL_ifaddr CL_getifaddr(CL_status&);
   CL_ifname CL_getifname(CL_status&);
   CL_ifaddr CL_getnodeaddr(CL_status&);
   CL_nodename CL_getnodenamebyif(CL_status&);
   CL_status CL_isavail();
   CL_netif& operator=(const struct cl_netif&);
};
```

Node Object Class

Node class data and member functions:

```
class CL_node  {
   public:
   CL_clusterid cln_clusterid;   // Cluster Id
   CL_nodeid cln_nodeid;         // Cluster Node Id (internal)
   CL_nodename cln_nodename;     // Cluster Node name
   CL_state cln_state;           // Cluster Node State
   int cln_nif;                  // Cluster Node Number of
                                 // Interfaces
   CL_netif *cln_if;             // Cluster Node
                                 // interfaces
   CL_cluster *cln_pcluster;     // pointer to parent cluster
                                 // object
   CL_route CL_bestroute(CL_status&);
   CL_node CL_getinfo(CL_status&);
   CL_status CL_isavail();
   CL_node& operator=(const struct cl_node&);
};
```

Note: The pointers to parent objects included in the object class data are provided in case you want to set up a tree structure of objects. These pointers are not filled by the Clinfo C++ API.

Resource Group Object Class

Resource Group data and member functions:

```
class CL_group  {
    public:
    CL_clusterid clg_clusterid;         // Cluster Id
    CL_groupid clg_group_id             // Resource Group Id
    CL_groupname clg_name;              // Resource group name

    /* The following field is deprecated in HACMP 5.2 and will not be
     * used. The data field itself is not removed from the data
     * structures to maintain the backward compatibility */

    CL_rg_policies clg_policy;          // Resource Group Policy
    CL_rg_policies clg_startup_policy;
    CL_rg_policies clg_fallover_policy;
    CL_rg_policies clg_fallback_policy;
    CL_rg_policies clg_site_policy;     // Resource Group site policy
    CL_user_policy_name clg_user_policy_name;
        // User defined policy

    int clg_num_nodes;
    int clg_node_ids[MAXNODES];         // Node ids in this group

    CL_resource_states clg_node_states[MAXNODES];
       //and their state
    CL_cluster *cln_pcluster;
       // pointer to parent cluster object
    CL_group CL_getinfo(CL_status&);
    CL_group& operator=(const struct cl_group&);
};
```

Note: The pointers to parent objects included in the object class data are provided in case you want to set up a tree structure of objects. These pointers are not filled by the Clinfo C++ API.

Functions Not Included in Any Object Class

The following functions are not included in any object class:

```
int CL_getallinfo (CL_cluster *, CL_status&);
```

This function is not a member function of class **CL_cluster**, since it returns the number of clusters, as well as information about all clusters rather than a particular cluster object.

```
CL_node CL_getlocalid(CL_status&);
```

This function is not a member function of class **CL_node**, since it returns information about the local host.

Assigning Class CL_netif data: cli_addr and cli_name

The following code illustrates how to assign network names and addresses. These assignments are used in the examples included on the reference pages in this chapter.

To assign a `cli_addr`, given `char *addr = "1.2.3.4"`:

```
netif.cli_addr.sin_family = AF_INET;
netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
```

To assign a `cli_name`:

```
strcpy(netif.cli_name.name, "node_name");
```

Overloaded Assignment Operator

The = assignment operators for the classes **CL_cluster**, **CL_netif**, **CL_group**, and **CL_node**, are overloaded in order to assign the C structures **cl_cluster**, **cl_netif**, **cl_group**, and **cl_node** to their corresponding C++ CL_ classes.

Functions Called Using the Clinfo C API

The following functions have not been translated from C to C++. You must call them using the Clinfo C API:

Event Functions

```
int cl_registereventnotify(int, struct cli_enr_req_t *);
int cl_unregistereventnotify(int, struct cli_enr_req_t *);
int cl_getevent(cli_en_msg_t *);
```

Utility Functions

```
void cl_perror(int, char *);
char *cl_errmsg (int status);
    /*for single-threaded applications*/
char *cl_errmsg_r(int status, char cbuf);
    /*for multi-threaded applications*/
```

Upgrading Applications from Earlier Clinfo Releases

Earlier releases of Clinfo used integers to identify cluster nodes (node IDs) instead of character strings (node names).

The following sections give you some examples of how to convert calls in your application to various Clinfo C++ API routines that previously used a *nodeid* parameter. For each routine, an example of its use with node IDs is followed by an example using node names.

CL_getlocalid

This is an example of the **CL_getlocalid** routine from an earlier release that uses node ID:

```
CL_status s;
CL_node lnode;lnode = node.CL_getlocalid(s);
if (s < 0)
   cl_errmsg(s);
printf("cluster id = %d, node id = %d", lnode.cln_clusterid,
   lnode.cln_nodeid);
```

In the new version of the example of the C++ API **CL_getlocalid** routine, note the change in the **printf** statement.

```
// This function is not a member of a class.

CL_status s;
CL_node lnode;
char cbuf[CL_ERRMSG_LEN];
lnode = CL_getlocalid(s);
if (s < 0)
   cl_errmsg_r(s, cbuf);
printf("cluster id = %d, node name = %s", lnode.cln_clusterid,
   lnode.cln_nodename.name);
```

CL_isavail

The following is an example of the **CL_isavail** routine from an earlier release that uses node ID:

```
CL_status s;
CL_netif netif;
netif.cli_clusterid = 2;
netif.cli_nodeid = 2;
netif.cli_addr.sin_family = AF_INET;
netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
s = netif.CL_isavail();

printf("status = %d", s);
```

The new version of the example of the C++ API **CL_isavail** routine changes the assignment statement, since the previous version used *cli_nodeid*; now it is *cln_name.name*.

```
CL_status s;
CL_netif netif;
netif.cli_clusterid = 2;
strcpy(netif.cln_name.name, "moby");
netif.cli_addr.sin_family = AF_INET;
netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
s = netif.CL_isavail();

printf("status = %d", s);
```

CL_getprimary

The following is an example of the **CL_getprimary** routine from an earlier release that uses node ID:

```
CL_cluster clus;
CL_status s;
CL_nodeid nid;
clus.clc_clusterid = 2;
nid = clus.CL_getprimary(s);
if (s < 0)
   cl_errmsg(s);
printf("nodeid = %d", nid);
```

The new version of the example of the C++ API **CL_getprimary** routine changes because a primary node is now recognized by its name (a string) rather than by an integer.

```
CL_clusterid clusterid;
CL_cluster clus;
CL_status status;
CL_nodename name;
CL_node node;
char cbuf[CL_ERRMSG_LEN];
clus.clc_clusterid = 2;
status = clus.CL_getprimary(cluster.clc_clusterid);
if (status < 0)
   cl_errmsg_r(status, cbuf);
printf( "Cluster %d's primary node is %s",
   cluster.clc_clusterid, cluster.clc_primary.name);
```

Requests

The Clinfo C++ API has the following types of requests:

- Cluster Information Requests
- Node Information Requests
- Network Interface Information Requests
- Resource Group Information Requests.

Cluster Information Requests

The following cluster information requests return information about a cluster:

CL_getallinfo	Calls the C function **cl_getnodemap.** This request is a member function associated with the **cl_cluster** class. The request returns information about all nodes in a cluster, given a cluster ID. A pointer to an array of node objects is passed as a reference argument to the function.
CL_getallinfo	Calls the C function **cl_getclusters**. Returns the number of clusters found. A pointer to an array of cluster objects is passed as a reference argument to the function. (When **CL_getallinfo** is called to get info on all clusters, the function is not a member function of the class **CL_cluster**.)
CL_getclusterid	Calls the C function **cl_getclusterid**, returns the cluster ID given a cluster name.
CL_getinfo	Calls the C function **cl_getcluster**. Returns information about the cluster with the specified cluster ID.
CL_getprimary	Calls the C function **cl_getprimary**. Returns the node name of the user-defined primary Cluster Manager in the specified cluster.
CL_isavail	Calls the C function **cl_isclusteravail**. Returns the status of the cluster with the specified cluster ID.
CL_getgroupinfo	Calls the C function **cl_getgroupmap**. Returns the number of resource groups found. A pointer to an array of resource group objects is passed as a reference object.

Node Information Requests

The following node information requests return information about the nodes in the cluster:

CL_bestroute	Calls the C function **cl_bestroute**. Returns the local/remote IP address pair that offers the most direct route to the specified node.
CL_getinfo	Calls the C function **cl_getnode**. Returns information about the node specified by a cluster ID/node name pair.
CL_getlocalid	Calls the C function **cl_getlocalid**. Returns a **CL_node** object that contains the cluster ID and node name of the host making the request. (This function is not a member function of the class **CL_node**.)
CL_isavail	Calls the C function **cl_isnodeavail**. Returns the status of the specified node, given its cluster ID and node name.

Network Interface Information Requests

The following network interface information requests return information about the interfaces connected to a node:

CL_getclusterid	Calls the C function **cl_getclusteridbyifaddr** or **cl_getclusteridbyifname**. Returns the cluster's network interface name if the address is specified; returns its network interface address if the name is specified.
CL_getifaddr	Calls the C function **cl_getifaddr**. Returns the network interface address of the interface with the specified cluster ID and network interface name.
CL_getifname	Calls the C function **cl_getifname**. Returns the interface name of the interface with the specified cluster ID and network interface address.
CL_getnodeaddr	Calls the C function **cl_getnodeaddr**. Returns the network interface address associated with the specified cluster ID and network interface name.
CL_getnodenamebyif	Calls the C function **cl_getnodenamebyifaddr** or **cl_getnodenamebyifname**. Returns the node name by calling **cl_getnodenamebyifaddr** if the interface address is known, or by calling **cl_getnodenamebyifname** otherwise.
CL_isavail	Calls the C function **cl_isaddravail**. Returns the status of the specified network interface, given its cluster ID, node name, and network interface address.

Resource Group Information Requests

The following resource group information request returns information about cluster resource groups:

CL_getinfo	Returns all information about the specific resource group in the specified cluster.

Event Notification Requests

The following event notification routines return information about cluster, node, or network events. You must call these routines from the Clinfo C API.

cl_getevent	Gets information when an event signal is received, and returns an event notification message received from Clinfo.
cl_registereventnotify	Registers a list of event notification requests with Clinfo, and returns a signal to the calling process when a registered event occurs.
cl_unregistereventnotify	Unregisters a list of event notification requests with Clinfo.

CL_getallinfo Routine

Syntax

```
int CL_getallinfo (CL_cluster *clusters, CL_status s)
```

Description

Returns information about known clusters.

Required Input Object Data

None.

Return Value

***clusters**	A pointer to an array of cluster objects is passed as a reference argument to the function.
int	The number of clusters found.

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.

Example

```
CL_status status;
CL_cluster clusters[8];
CL_cluster *ret = &clusters[0];
int numclus;

numclus = CL_getallinfo(ret, status);

if (status < 0) {
       cl_perror(status, progname);
} else {
      printf("number of clusters found: %d\n", numclus);
      for (int i=0; i<numclus; i++) {
              printf("[%d] cl %d", i, ret[i].clc_clusterid);
              printf(" st %d", ret[i].clc_state);
              printf(" su %d", ret[i].clc_substate);
              printf(" pr %s", ret[i].clc_primary.name);
              printf(" na %s\n", ret[i].clc_name.name);
      }
}
```

CL_getlocalid Routine

Syntax

```
CL_node CL_getlocalid (CL_status s)
```

Description

Returns a **CL_node** object that contains the cluster ID and the node name of the host making the request. This request returns an error status on nodes not currently active in the cluster.

Required Input Object Data

None.

Return Value

CL_node	Node object with local cluster and node name.

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVNODE	Node is not a cluster node.

Example

This example uses the **cl_errmsg** routine to illustrate proper programming for a single-threaded application. If your program is multi-threaded, you must use the **cl_errmsg_r** routine.

```
CL_status status;
CL_node lnode;

lnode = CL_getlocalid(status);
if (status < 0) {
    cl_errmsg(status);
} else {
    printf("cluster id = %d, node name = %s\n", lnode.cln_clusterid,
        lnode.cln_nodename.name);
}
```

CL_cluster::CL_getallinfo Routine

Syntax

```
int *CL_cluster::CL_getallinfo (CL_node *nodes, CL_status s)
```

Description

Returns information about all the nodes in a cluster.

Required Input Object Data

CL_cluster::clc_clusterid	The cluster ID of the desired cluster.

Return Value

nodes	A pointer to an array of node objects is passed as a reference argument to the function.
int	The number of nodes in the cluster.

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

```
CL_cluster cluster;
CL_status status;
CL_node nodes[8];
CL_node *ret = &nodes[0];
int numnodes;

cluster.clc_clusterid = 1113325332;
numnodes = cluster.CL_getallinfo(ret, status);
if (status < 0) {
  cl_perror(status, progname);
} else {
  printf("number of nodes in cluster: %d\n", numnodes);
  for (int i=0; i<numnodes; i++) {
      printf("[%d] clusterid %d", i, ret[i].cln_clusterid);
      printf(" nodeid %d", ret[i].cln_nodeid);
      printf(" state %d", ret[i].cln_state);
      printf(" interfaces %d\n", ret[i].cln_nif);
  }
}
```

CL_cluster::CL_getclusterid Routine

Syntax

```
CL_clusterid CL_cluster::CL_getclusterid(CL_status s)
```

Description

The **CL_getclusterid** routine returns the cluster ID of the cluster with the specified name.

Required Input Object Data

CL_cluster::clc_name	Name of target cluster.

Return Value

CL_clusterid	The desired cluster ID.

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERNAME	The request specified an invalid cluster name.

Example

```
    CL_cluster cluster;
    CL_status status;
    CL_clusterid clusterid;
    char cbuf[CL_ERRMSG_LEN];

    strcpy(cluster.clc_name.name, "site1");
    clusterid = cluster.CL_getclusterid(status);
    if (status < 0) {
      cl_errmsg(status);
    } else {
      printf("clusterid = %d\n", clusterid);
    }
}
```

CL_cluster::CL_getinfo Routine

Syntax

```
CL_cluster CL_cluster::CL_getinfo (CL_status s)
```

Description

The **CL_getinfo** routine returns information about the cluster specified by the cluster ID.

Required Input Object Data

CL_cluster::clc_clusterid	ID of target cluster.

Return Value

CL_cluster	Information about the specified cluster.

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

```
CL_cluster cluster;
CL_status status;
CL_cluster ret;
char cbuf[CL_ERRMSG_LEN];

cluster.clc_clusterid = 1113325332;
ret = cluster.CL_getinfo(status);
if (status < 0) {
  cl_errmsg(status);
} else {
  printf("clusterid %d ", ret.clc_clusterid);
  printf("state %d ", ret.clc_state);
  printf("substate %d ", ret.clc_substate);
  printf("primary %s ", ret.clc_primary.name);
  printf("name %s\n", ret.clc_name.name);
}
```

CL_cluster::CL_getprimary Routine

Syntax

```
CL_nodename CL_cluster::CL_getprimary (CL_status s)
```

Description

Returns the node name of the user-designated primary Cluster Manager for the specified cluster.

Required Input Object Data

CL_cluster::clc_clusterid	The cluster ID for which the primary ID is desired.

Return Value

CL_nodename	The node name of the primary Cluster Manager.

Status Value

CL_status status	Status passed by reference. Output parameter for the return codes.
CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_NOPRIMARY	No primary information is available. This status usually indicates that the user has not designated a primary cluster manager.
CLE_IVCLUSTER	The cluster is not available.

Example

```
CL_cluster cluster;
CL_status status;
CL_nodename name;

cluster.clc_clusterid = 1;
name = cluster.CL_getprimary(status);
if (status < 0) {
    cl_errmsg(status);
} else {
    printf( "cluster %d's primary node is %s\n",
                    cluster.clc_clusterid, cluster.clc_primary.name);
}
```

CL_cluster::CL_isavail Routine

Syntax

```
CL_status CL_cluster::CL_isavail()
```

Description

Returns the status code CLE_OK if the specified cluster is available.

Required Input Object Data

CL_cluster::clc_clusterid	The cluster ID of the desired cluster.

Return Value

CL_status	Status of the specified cluster.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_IVCLUSTER	The request specified an invalid cluster.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

```
CL_status status;
  CL_cluster cluster;

  cluster.clc_clusterid = 1113325332;
  status = cluster.CL_isavail();
  printf("status = %d\n", status);
```

CL_cluster::CL_getgroupinfo Routine

Syntax

```
int CL_cluster::CL_getgroupinfo (CL_group *groups, CL_status&);
```

Description

Returns information about all the resource groups in the cluster.

Required Input Object Data

CL_cluster::clc_clusterid	The cluster ID of the desired cluster.

Return Value

groups	A pointer to a group of resource group objects is passed as a reference argument to the function.
int	The number of resource groups in the cluster

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

```
CL_status status;
CL_cluster cluster;
CL_group groups[MAXGROUPS];
int numgroups;

cluster.clc_clusterid = 1113325332;
numgroups = cluster.CL_getgroupinfo(&groups[0], status);
if (status < 0) {
  cl_perror(status, progname);
} else {
  printf("There are %d groups in cluster %d\n", numgroups,
  cluster.clc_clusterid);
  for (int i=0; i<numgroups; i++) {
    printf("Group %d is id %d\n", i, groups[i].clg_group_id);
    printf("Group %d is %s and has %d nodes\n",
    i, groups[i].clg_name.name, groups[i].clg_num_nodes);
  }
}
```

CL_group::CL_getinfo Routine

Syntax

```
CL_Group::CL_getinfo (CL_status s);
```

Description

Returns a group object that contains information about the group, given a group object with the cluster ID and group name.

Required Input Object Data

CL_group::clg_clusterid	Cluster ID of the desired resource group.
CL_group::clg_name.name	Name of the resource group.

Return Value

CL_group	The desired resource group and its full information.

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.

Example

```
CL_status status;
CL_group group;
CL_group ret;

group.clg_clusterid = 1113325332;
strcpy(group.clg_name.name, "rg01");
ret = group.CL_getinfo(status);
if (status < 0) {
  cl_perror(status, progname);
} else {
  printf("There are %d nodes in group %s\n",
  ret.clg_num_nodes, ret.clg_name.name);
}
```

CL_netif::CL_getclusterid Routine

Syntax

```
CL_clusterid CL_netif::CL_getclusterid (CL_status s)
```

Description

Returns the name of the cluster with the specified network interface address. Alternatively, returns the network interface address of a cluster given its network interface name.

Required Input Object Data

CL_netif::cli_addr	The network interface address whose cluster ID is desired.

or

CL_netif::cli_name	The network interface name whose cluster ID is desired.

Return Value

CL_clusterid	The cluster ID (a non-negative integer).

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVADDRESS	The request specified an invalid network interface address.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

```
// example using interface name

CL_status status;
CL_clusterid clid;
CL_netif netif;

strcpy(netif.cli_name.name, "geotest9");
netif.cli_addr.sin_addr.s_addr = NULL;
clid = netif.CL_getclusterid(status);
if (status < 0) {
  cl_errmsg(status);
} else {
  printf("clusterid = %d\n", clid);
}

// example using interface address

  CL_status status;
  CL_clusterid clusterid;
  CL_netif netif;
  char *addr = "1.1.1.7";

  netif.cli_addr.sin_family = AF_INET;
  netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
  netif.cli_name.name[0] = NULL;
  clusterid = netif.CL_getclusterid(status);
  if (status < 0) {
       cl_errmsg(status);
  } else {
       printf("clusterid = %d\n", clusterid);
  }
```

CL_netif::CL_getifaddr Routine

Syntax

```
CL_ifaddr CL_netif::CL_getifaddr (CL_status s)
```

Description

Returns the network interface address of the interface with the specified cluster ID and network interface name.

Required Input Object Data

CL_netif::cli_clusterid	The cluster ID of the desired network interface.
CL_netif::cli_name	The name of the desired network interface.

Return Value

CL_ifaddr	The network interface address desired.

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. Usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

```
CL_status status;
CL_ifaddr ifaddr;
CL_netif netif;
char cbuf[CL_ERRMSG_LEN];
netif.cli_clusterid = 1113325332;
strcpy(netif.cli_name.name, "geotest9");
ifaddr = netif.CL_getifaddr(status);
if (status < 0) {
  cl_errmsg(status);
} else {
  printf("ifaddr = %s\n", inet_ntoa(ifaddr.sin_addr));
}
```

CL_netif::CL_getifname Routine

Syntax

```
CL_ifname CL_netif::CL_getifname (CL_status s)
```

Description

Returns the network interface name, given a cluster ID and network interface address; or, given a cluster ID and node name, returns a network interface name.

If the request specifies a **cli_addr** parameter, Clinfo examines the network portion of the address and seeks an interface on the same network. If a match is found, the **CL_getifname** routine returns the name associated with that interface.

If the **cli_addr** parameter is NULL, Clinfo decides which interface on the specified node is most easily accessed from the local host, and the **CL_getifname** routine returns the name associated with that interface. If both interfaces are equally accessible from the local node, Clinfo chooses one and returns the name associated with that interface.

In all cases, the **CL_getifname** routine returns the name as a NULL-terminated string.

Required Input Object Data

CL_netif::cli_clusterid, cli_addr	The cluster ID of the desired network interface and the network interface address.

or

CL_netif::cli_clusterid, cli_nodename	The cluster ID and node name of the desired network interface.

Return Value

CL_ifname	The network interface name.

Status Values

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVADDRESS	The request specified an invalid network interface address.
CLE_IVNODENAME	The request specified an invalid node name.

Example

```
// CL_netif::CL_getifname get interfacename given clusterid and nodename

  CL_status status;
  char cbuf[CL_ERRMSG_LEN];
  CL_ifname ifname;
  CL_netif netif;

  netif.cli_addr.sin_addr.s_addr = NULL;
  netif.cli_clusterid = 1113325332;
  strcpy (netif.cli_nodename.name, "node1");
  ifname = netif.CL_getifname(status);
  if (status < 0) {
     cl_errmsg(status);
  } else {
     printf("ifname = %s\n", ifname.name);
  }
```

CL_netif::CL_getnodeaddr Routine

Syntax

```
CL_ifaddr CL_netif::CL_getnodeaddr(CL_status s)
```

Description

Returns the IP address associated with the specified cluster ID and network interface name.

Required Input Object Data

CL_netif::cli_clusterid	The cluster ID of the desired network interface.
CL_netif::cli_name	The name of the desired network interface.

Return Value

CL_ifaddr	Network interface address.

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

```
CL_status status;
CL_netif netif;
CL_ifaddr ifaddr;
char cbuf[CL_ERRMSG_LEN];

netif.cli_clusterid = 1113325332;
strcpy(netif.cli_name.name, "geotest9");
ifaddr = netif.CL_getnodeaddr(status);
if (status < 0) {
  cl_errmsg(status);
} else {
  printf("ifaddr = %s\n", inet_ntoa(ifaddr.sin_addr));
}
```

CL_netif::CL_getnodenamebyif Routine

Syntax

```
CL_nodename CL_netif::CL_getnodenamebyif (CL_status s)
```

Description

Returns the node name when given either a cluster ID and a network interface address or a cluster ID and a network interface name.

If the network interface address is specified and **cli_name** is NULL, then **cli_name** is returned. Conversely, if **cli_name** is given and **cli_addr** is NULL, **cli_addr** is returned. If both **cli_name** and **cli_addr** are non-NULL, **cli_addr** takes precedence. If both are NULL, the error code CLE_ BADARGS is returned.

Note: This routine replaces the **CL_getnodename** routine, which was available in previous releases.

Required Input Object Data

CL_netif::cli_clusterid, cli_addr	The cluster ID of the desired node, and the network interface address of the node.

or

CL_netif::cli_clusterid, cli_name	The cluster ID of the desired node, and the name of the network interface.

Return Value

CL_nodename	The node name.

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVADDRESS	The request specified an invalid network interface address.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

```
CL_status status;
char cbuf[CL_ERRMSG_LEN];
CL_nodename nname;
CL_netif netif;
char *addr = "9.57.28.23";

netif.cli_clusterid = 1113325332;
netif.cli_addr.sin_family = AF_INET;
netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
netif.cli_name.name[0] = NULL;
nname = netif.CL_getnodenamebyif(status);
if (status < 0) {
   cl_errmsg(status);
} else {
   printf("node name = %s\n", nname.name);
}
```

CL_netif::CL_isavail Routine

Syntax

```
CL_status CL_netif::CL_isavail()
```

Description

Returns the status code CLE_OK if the specified network interface is available.

Required Input Object Data

CL_netif::cli_clusterid	The cluster ID of the desired network interface.
CL_netif::cli_nodename	The node name of the desired network interface.
CL_netif::cli_addr	The address of the desired network interface.

Return Value

CL_status	Status of the specified network interface.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.
CLE_IVADDRESS	The request specified an invalid interface address.
CLE_IVNETIF	The network interface is not available.

Example

```
CL_status status;
CL_netif netif;
char *addr = "9.57.28.23";

netif.cli_clusterid = 1113325332;
strcpy(netif.cli_name.name, "geotest9");
netif.cli_addr.sin_family = AF_INET;
netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
strcpy(netif.cli_nodename.name, "node1");
status = netif.CL_isavail();
if (status < 0) {
     cl_perror(status,"netif.CL_isavail failed");
}
printf("status = %d\n", status);
```

CL_node::CL_bestroute Routine

Syntax

```
CL_route CL_node::CL_bestroute(CL_status s)
```

Description

The **CL_bestroute** routine returns the local/remote IP address pair for the most direct route to the node specified in the object.

The route returned by the **CL_bestroute** routine depends on the node making the request. Clinfo first builds a list of all working network interfaces on the local node, and then compares this list to the available interfaces on the specified node. The routine first compares HACMP-defined private interfaces (such as a serial optical channel) to local interfaces. If no match is found, the routine then compares HACMP-defined public interfaces to local interfaces. If there is still no match, the routine selects the first defined interface on the local node and the first defined interface on the remote node.

If a pair of local and remote interfaces exist that are on the same network, they are returned in **CL_route**. Otherwise, an interface on the specified node is chosen as the remote interface, and the primary local interface is returned as the local end of the route.

Required Input Object Data

CL_node::cln_clusterid, cln_nodename Cluster ID and node name of target node.

Return Value

CL_route	Local/remote IP address pair that indicates the most direct route to the specified node.

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_OK	The request completed successfully.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_NOROUTE	No route available.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.

Example

```
CL_status status;
CL_node node;
CL_route route;
char cbuf[CL_ERRMSG_LEN];

node.cln_clusterid = 1113325332;
strcpy(node.cln_nodename.name, "node1");
route = node.CL_bestroute(status);
if (status < 0) {
  cl_errmsg(status);
} else {
  // don't call inet_ntoa twice in one printf!
  printf("local = %s  ", inet_ntoa(route.localaddr.sin_addr));
  printf("remote = %s\n", inet_ntoa(route.remoteaddr.sin_addr));
}
```

CL_node::CL_getinfo Routine

Syntax

```
CL_node CL_node::CL_getinfo(CL_status s)
```

Description

Returns a node object that contains information about the node, given a node object with a cluster ID and node name.

Required Input Object Data

CL_node::cln_clusterid	Cluster ID of target node.
CL_node::cln_nodename	Node name of target node.

Return Value

CL_node	The desired node and its information.

Status Value

CL_status s	Status passed by reference. Output parameter that holds the return code.
CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_BADARGS	Missing or invalid parameters. This status usually indicates that a NULL pointer was specified for an output parameter address.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.

Example

```
CL_status status;
CL_node node;
CL_node ret;
char cbuf[CL_ERRMSG_LEN];

node.cln_clusterid = 1113325332;
strcpy(node.cln_nodename.name, "node1");
ret = node.CL_getinfo(status);
if (status < 0) {
    cl_errmsg(status);
} else {
    printf("clusterid %d  ", ret.cln_clusterid);
    printf("nodename %s  ", ret.cln_nodename.name);
    printf("state %d  ", ret.cln_state);
    printf("nif %d\n", ret.cln_nif);
}
```

CL_node::CL_isavail Routine

Syntax

```
CL_status CL_node::CL_isavail()
```

Description

Returns the status code CLE_OK if the specified node is available.

Required Input Object Data

CL_node::cln_clusterid	The cluster ID of the desired node.
CL_node::cln_nodename	The node name of the desired node.

Return Value

CL_status	Status of the specified node.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable **errno** for additional information.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.
CLE_IVNODE	The node is not available.

Example

```
CL_status status;
CL_node node;

node.cln_clusterid = 1113325332;
strcpy(node.cln_nodename.name, "node1");
status = node.CL_isavail();
printf("status = %d\n", status);
```

Chapter 4: Sample Clinfo Client Program

This chapter lists a sample **clinfo.rc** script and the source code of a C program called from that script. This program reports the status of each service network interface on a given cluster node, and of the node itself.

Overview

The sample Clinfo client application program, **cl_status**, is shown here within the context of a typical customized **clinfo.rc** script. **clinfo.rc** is the HACMP for AIX script run by Clinfo following cluster topology changes. The **clinfo.rc** script is listed first, followed by the **cl_status** program. The script and the program are commented to explain their usage.

Sample Customized clinfo.rc Script

```
#!/bin/ksh
##############################################################################
# Filename: /usr/sbin/cluster/etc/clinfo.rc
#
# Description: clinfo.rc is run by clinfo on clients following cluster
#                 topology changes. This particular example demonstrates
#                 user process management for a highly available database in a
#                 two-node primary/standby configuration. Most database
#                 client programs are state-dependent, and require
                  #restart following a node failure. This example provides
#                 user notification and application shutdown during
#                 appropriate topology changes.
#
#############################################################################
##############################################################################
# Grab Parameters Passed
##############################################################################
EVENT=$1                   # action, one of {join, fail, swap}
INTERFACE=$2               # target address label
CLUSTERNAME="cluster1"     # cluster name
NODENAME="victor"          # primary node name
WATCHIF="svc_en0"          # interface to monitor

##############################################################################
#   Name:  _arp_flush
#   This function flushes the entire arp cache.
#   Arguments:  none
#   Return Value:  none
##############################################################################
_arp_flush()
{
  for IPADDR in $(/etc/arp -a |/bin/sed -e 's/^.*(.*).*$//' -e /incomplete/d)
    do
        /etc/arp -d $IPADDR
    done
}

##############################################################################
```

```
#
#   Name:  _kill_user_procs
#
#   This function kills user processes associated with the specified
#   interface.
#
#   Arguments:  interface
#   Return Value:  none
##########################################################################
_kill_user_procs()
{
    print _kill_user_procs
  # place commands appropriate to the database in use here

}
# The main if statement disregards status changes for all interfaces except
# WATCHIF, which in this example is svc_en0.
if [[ "$INTERFACE" = "WATCHIF" ]]
then
  case "$EVENT" in
    "join") # interface label $INTERFACE has joined the cluster
            # perform necessary activity here, such as user notification, restoration
of user access,
          # and arp cache flushing.
            exit 0
            ;;

    "fail")# Use api calls in cl_status to determine if interface
          # failure is a result of node failure.

  CLSTAT_MSG=$(cl_status $CLUSTERNAME $NODENAME)
            CLSTAT_RETURN=$?  # return code from cl_status

  case "$CLSTAT_RETURN" in
              0)  # Node UP
                  # Notify users of application availability
                  wall "Primary database is now available."
                  # flush arp cache
                  _arp_flush
                 ;;

              1)  # Node DOWN
                  # Notify users of topology change and restart requirement
                  touch /etc/nologin    # prevent new logins
                  wall "Primary database node failure. Please login again
                           2 minutes"
           sleep 10
                  # Kill all processes attached to WATCHIF interface
                  _kill_user_procs $WATCHIF
                  # flush arp cache
                 _arp_flush
                  rm -f /etc/nologin     # enable logins
                  ;;

              *) # Indeterminate node state
                 # flush arp cache
                  _arp_flush
                 exit 1
                 ;;

            esac  # case $CLSTAT_RETURN
            ;;
    "swap")# interface has been swapped
          # flush arp cache.
          _arp_flush
          ;;
  esac    # case $EVENT
else
    # event handling for other interfaces here, if desired
    /bin/true
fi
```

cl_status.c

```
/*
 * Program:  cl_status.c
 *
 * Purpose:  For systems running the clinfo daemon as a client, cl_status
 *           will determine if the node for the network interface passed
 *           to it is active in the cluster.
 *
 * Usage:    [path/]cl_status clustername nodename
 *
 * Returns:  0 = Node up
 *           1 = Node down
 *           2 = ERROR - Status Unavailable
 *
 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <cluster/clinfo.h>
#include <strings.h>

void usage()
{
  printf("usage: cl_status clustername nodename");
  printf("Returns status of node in HACMP cluster.");
}

int main(int argc, char *argv[])
{
  int clusterid, node_status;
  char *clustername, *nodename;

  if(argc < 3)
  {
    /* incorrect syntax to cl_status call */
    usage();
    exit(2);
  }
  clustername = strdup(argv[1]);
  if (strlen(clustername) > CL_MAXNAMELEN)
  {
    printf("error: clustername exceeds maximum length of %i characters",
     CL_MAXNAMELEN);
    exit(2);
  }
  nodename = strdup(argv[2]);
  if (strlen(nodename) > CL_MAXNAMELEN)
  {
    printf("error: nodename exceeds maximum length of %i characters",
     CL_MAXNAMELEN);
    exit(2);
  }
  /* convert clustername (string) to clusterid (non-negative integer) */
  clusterid = cl_getclusterid(clustername);
  switch(clusterid)
  {
    case CLE_SYSERR: perror("system error");
                exit(5);
                break;

    case CLE_NOCLINFO: cl_perror(clusterid, "error");
                exit(5);
                break;

    case CLE_BADARGS:
    case CLE_IVCLUSTERNAME: /* typically a usage error */
                cl_perror(clusterid, "error");
                usage();
                exit(2);

    default: /* valid clusterid returned */
              ;
  }
  node_status = cl_isnodeavail(clusterid, nodename);
  switch (node_status)
  {
    case CLE_OK: /* Node up */
                printf("node %s up", nodename);
                exit(0);
                break;
    case CLE_IVNODENAME:  /* "Illegal node name" */
                cl_perror(node_status, "node unavailable");
                exit(2);
                break;

    default:
                cl_perror(node_status, "node unavailable");
                exit(1);
  }
}
```

Appendix A: Implementation Specifics

There are two key components to Clinfo: the **clinfo** daemon and the API library.

The **clinfo** daemon is an SNMP-based monitor. SNMP is a industry-wide set of standards for monitoring and managing TCP/IP-based networks. SNMP includes a protocol, a database specification, and sets of data objects.

The sets of data objects form a Management Information Base (MIB). SNMP provides a standard MIB that includes information such as IP addresses and the number of active TCP connections. The actual MIB definitions are encoded into the agents running on a system. The standard SNMP agent in AIX is the SNMP daemon, **snmpd**.

Programmers use SNMP operations to implement programs that will monitor and manage networks. These programs can receive information about the state of a network from **snmpd**, and pass the information on to clients and applications.

SNMP can be extended using the SNMP Multiplexing (SMUX) protocol to include *enterprise-specific* MIBs that contain information relating to a discrete environment or application. A management agent (a SMUX peer daemon) retrieves and maintains information about the objects defined in its MIB, and makes this information available to a specialized network monitor or network management station.

The HACMP software provides this SMUX peer function through the Cluster manager daemon. The **clinfo** daemon retrieves this information from the HACMP MIB (and indirectly) through the Cluster Manager.

The Clinfo API library (in all its variations) interacts with the **clinfo** daemon to provide access to the cluster information. Although the same information is available directly through SNMP, the Clinfo library provides a greatly simplified programming model allowing client programs to avoid the complexities of the SNMP API. The Clinfo API provides routines to retrieve all information related to cluster entities like nodes or resource groups (SNMP requires you to fetch these items one at a time) and routines to register for specific cluster events (SNMP requires traps to implement similar function). Variations of the library—C, C++, thread safe, and so forth—provide a consistent model for a variety of runtime environments.

The **clinfo** daemon and library can run on HACMP cluster nodes or on a non-cluster node, provided the daemon can access SNMP on a cluster node through TCP/IP.

Note: HACMP 5.3 integrates the functionality of the SMUX Peer daemon (**clsmuxpd**) into the **Cluster Manager**. This integrated function eliminates the **clsmuxpd** daemon.

The following sections provide additional details on how Clinfo operates in the HACMP environment:

- Cluster Manager and Clinfo
- SNMP Community Name and Clinfo.

Cluster Manager and Clinfo

The Cluster Manager daemon (**clstrmgr)** is an HACMP subsystem that monitors a cluster and initiates recovery actions if necessary. The Cluster Manager reports on cluster behavior so that other programs can determine if changes have occurred within the cluster and if necessary, respond to those changes.

Once **the Cluster Manager** gets the cluster information, it maintains an updated topology of the cluster in the HACMP for AIX MIB, as it tracks events and resulting states of the cluster. Clinfo, running on a client machine or on a cluster node, queries **the MIB** for updated cluster information, and gives an application access to the HACMP for AIX MIB information, through an application programming interface.

By default, Clinfo periodically polls **SNMP processes** for updated information on events (every 15 seconds). It is possible to start Clinfo with an option (**-a**), which enables Clinfo to receive this information as soon as an event occurs. In this case, **the Cluster Manager** sends trap messages when it receives the event information. Clinfo then immediately queries **the MIB** for the event information instead of waiting for the next polling period.

Note: When Clinfo is started with the **-a** option, you cannot run NetView for AIX or any other application that expects to receive SNMP trap messages.

When Clinfo starts, it reads the **/usr/sbin/cluster/etc/clhosts** file. This file lists the service network interface IP address or IP label of all available nodes in each cluster of interest. Clinfo searches through this file for an active SNMP process on a node, starting at the first IP address in the **clhosts** file. Once it locates a SNMP process, Clinfo receives information about the topology and state of the cluster from that SNMP process.

If this connection is broken (the node goes down, for example), Clinfo tries to establish a connection to another node's SNMP process. Clinfo holds cluster topology information internally, in dynamically allocated data structures on the local node, once it has received cluster information from the **SNMP process** with which it first established communication. Therefore, it knows about other nodes in the cluster.

Note: HACMP 5.3 eliminates Clinfo shared memory. The **clinfo** daemon stores data internally and the client API retrieves data from the daemon through a UNIX domain socket connection.

The following figure shows the relationship between the Cluster Manager, Clinfo and the cluster nodes.

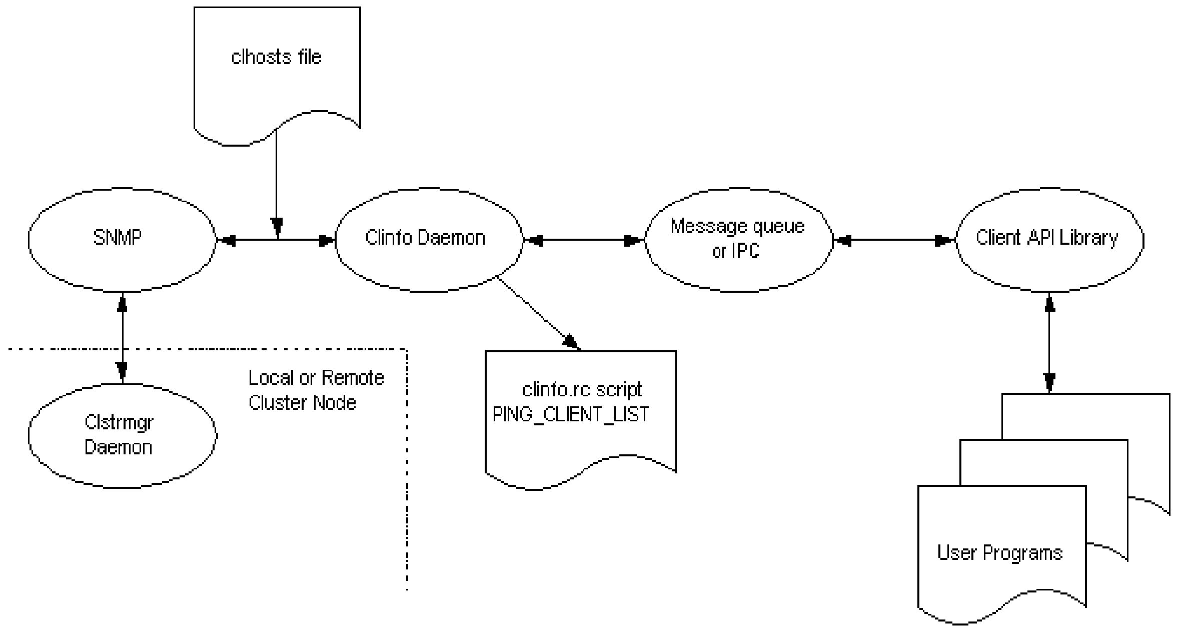

Figure 1. Cluster Manager and Clinfo Implementation

For Clinfo to work as expected, the **clhosts** file must contain the IP addresses of all HACMP server and client nodes to which Clinfo can communicate. The Clinfo daemon retrieves its information through SNMP from an HACMP server node—a node on which the Cluster Manager daemon (**clstrmgr**) is running. During startup, the **clinfo** daemon reads in the **clhosts** file to determine which nodes can communicate through SNMP as follows:

- For **clinfo** daemons running on the same server as the **clstrmgr** daemon, it reads in the local server-based **/usr/es/sbin/cluster/etc/clhosts** file, which only contains the IP address associated with the loopback address.
- For **clinfo** daemons running on client nodes, that is, nodes on which the **clstrmgr** daemon is not running, for highest availability, the client-based **/usr/es/sbin/cluster/etc/clhosts** file should contain the IP addresses of all of the HACMP server nodes. In this way, if a particular HACMP server node is unavailable (for example, powered down), then the **clinfo** daemon on the client node can attempt to connect to another HACMP server node through SNMP.

Note: For more information about the **clhosts** file see the *Concepts and Facilities Guide.*

If Clinfo does not succeed in communicating with a local **SNMP** process at startup, it does not get the cluster map and therefore cannot try to connect to another **SNMP** process.

SNMP Community Name and Clinfo

HACMP supports a SNMP Community Name other than “public”. If the default SNMP Community Name is changed in **/etc/snmpd.conf** to something different from the default of “public” HACMP functions correctly.

Note: The version of **/etc/snmpd.conf** depends on which version of AIX you are using. For AIX 5.2 or up, the default version used in HACMP is **snmpdv3.conf**.

The SNMP Community Name used by HACMP is the first name found that is not “private” or “system” using the **lssrc -ls snmpd** command. The Clinfo service gets the SNMP Community Name in the same manner.

The Clinfo service still supports the **-c** option for specifying SNMP Community Name but its use is not required. Using of the **-c** option is considered a security risk because running a **ps** command could find the SNMP Community Name.

WARNING: If it is important to keep the SNMP Community Name protected in Clinfo, change permissions on **/tmp/hacmp.out**, **/etc/snmpd.conf**, **/smit.log** and **/usr/tmp/snmpd.log** to not be world readable (for example, 600).

Notices for HACMP Programming Client Applications

Index