Bull

HACMP 4.4

Programming Client Applications

AIX

ORDER REFERENCE
86 A2 60KX 02

Bull

HACMP 4.4
Programming Client Applications

AIX

Software

August 2000

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
86 A2 60KX 02

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

Trademarks and Acknowledgements

We acknowledge the right of proprietors of trademarks mentioned in this book.

AIX® is a registered trademark of International Business Machines Corporation, and is being used under licence.

UNIX is a registered trademark in the United States of America and other countries licensed exclusively through the Open Group.

Year 2000

The product documented in this manual is Year 2000 Ready.

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 for AIX, Version 4.4 (HACMP for AIX) 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.

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.

Readers of this guide are expected to know the C or C++ programming language and should be familiar with networking concepts.

How To Use This Guide

Overview of Contents

This guide provides both conceptual and reference information. It contains the following chapters:

- Chapter 1, Cluster Information Program, explains the concepts the user needs to understand in order to use the Clinfo APIs in a highly available application. It includes general information about SNMP and the enterprise-specific MIB distributed with the HACMP for AIX, Version 4.4 software.
- Chapter 2, Clinfo C API, describes the specific routines available for the C programmer.
- Chapter 3, Clinfo C++ API, describes the specific routines available for the C++ programmer.
- Chapter 4, Sample Clinfo Client Program, supplies code for a client application that uses the Clinfo C API, within the context of a customized **clinfo.rc** script.
- Appendix A, HACMP for AIX MIB, lists the HACMP for AIX MIB and contains brief information about SNMP commands.

Highlighting

The following highlighting conventions are used in this book:

Italics	Identifies new terms or concepts.
Bold	Identifies routines, commands, keywords, files, directories, 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.

Related Publications

The following publications provide additional information about the HACMP for AIX software:

- *Release Notes* in **/usr/lpp/cluster/doc/release_notes** describe hardware and software requirements
- *HACMP for AIX, Version 4.4: Concepts and Facilities,* order number 86 A2 54KX 02
- *HACMP for AIX, Version 4.4: Planning Guide,* order number 86 A2 55KX 02
- *HACMP for AIX, Version 4.4: Installation Guide,* order number 86 A2 56KX 02
- *HACMP for AIX, Version 4.4: Administration Guide,* order number 86 A2 57KX 02
- *HACMP for AIX, Version 4.4: Troubleshooting Guide,* order number 86 A2 58KX 02
- *HACMP for AIX, Version 4.4: Programming Locking Applications,* order number 86 A2 59KX 02
- *HACMP for AIX, Version 4.4: Master Index and Glossary,* order number 86 A2 65KX 02
- *HACMP for AIX, Version 4.4: Enhanced Scalability Installation and Administration Guide,* Volumes I and II, order numbers 86 A2 62KX 02-02 and 86 A2 89KX 01

The AIX document set, as well as manuals accompanying machine and disk hardware, also provide relevant information.

Ordering Publications

To order additional copies of this guide, use order number 86 A2 60KX 02.

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 cluster. Clinfo is a cluster monitor based on the Simple Network Management Protocol (SNMP).

Overview of the Cluster Information Program

An HACMP cluster is dynamic and can undergo various transitions in its state over time. For example, a node can join or leave the cluster, or a standby adapter can replace a service adapter. Each of these changes affects the composition of the cluster. Because a cluster can change over time, 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.

Clinfo is an SNMP-based monitor. SNMP is an industry-standard set of standards for monitoring and managing TCP/IP-based networks. SNMP includes a protocol, a database specification, and a set of data objects. A set of data objects forms 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 is the SNMP daemon, **snmpd**.

Programmers use SNMP operations to implement network monitor and network management programs, which can receive information about the state of a network from the **snmpd**, and pass the information on to clients and applications.

SNMP can be extended through the use of 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 for AIX software provides the HACMP for AIX MIB, associated with and maintained by the HACMP for AIX management agent, the Cluster SMUX peer daemon (**clsmuxpd**). Clinfo retrieves this information from the HACMP for AIX MIB through the **clsmuxpd**.

Clinfo can run on cluster nodes and on HACMP for AIX client machines. It makes information about the state of an HACMP cluster and its components available to clients and applications. Clinfo and its associated application programming interfaces enable a developer to write applications that recognize and respond to changes within a cluster.

The following sections briefly explain how Clinfo operates in the HACMP for AIX environment:

- Cluster SMUX peer daemon and Clinfo
- The Clinfo C and C++ APIs
- Cluster events tracked by Clinfo

Cluster SMUX Peer Daemon and Clinfo

When **clsmuxpd** starts running on a cluster node, it registers with **snmpd**, then continually receives cluster information from the Cluster Manager (**clstrmgr**). The **clstrmgr** is an HACMP subsystem that monitors a cluster and initiates recovery actions if necessary. It reports on cluster behavior so that other programs can find out if changes have occurred within the cluster and, if necessary, respond to those changes.

Once **clsmuxpd** gets the cluster information, it maintains an updated topology map 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 **clsmuxpd** 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 **clsmuxpd** for updated information on events (every 15 seconds). It is possible to start Clinfo with an option (**-a**) that enables Clinfo to receive this information as soon as an event occurs. In this case, **clsmuxpd** sends trap messages when it receives the event information from the **clstrmgr**. Clinfo then immediately queries **clsmuxpd** 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 configuration file **/usr/sbin/cluster/etc/clhosts**. The information in this file lists the IP address or IP label of the service adapter of at least one node in each cluster of interest. Clinfo searches for an active **clsmuxpd** process on a node, starting at the first IP address in the **clhosts** file. Once it locates a **clsmuxpd** process, Clinfo receives information about the topology and state of the cluster from that **clsmuxpd**.

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

For Clinfo to work as expected, the **clhosts** file must be properly configured. On a client machine, you must add the desired IP label or internet address of all HACMP for AIX nodes to which Clinfo may communicate. Note that this includes nodes from any cluster accessible through network connections.

As installed, the **clhosts** file on an HACMP for AIX node contains a loopback address. If Clinfo does not succeed in communicating with a local **clsmuxpd** at startup, it does not get the cluster map and therefore cannot try to connect to another **clsmuxpd.** For this reason, it is recommended that you replace the loopback address with all HACMP for AIX node addresses accessible through network connections with this node. The loopback address is provided only as a convenience.

Note: Clinfo has a **-c** option that lets the snmp community be set. You must use this option on any system where the default community (public) has been disabled.

Clinfo APIs

An application accesses HACMP MIB 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, Versions 4.1.1 and later include 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**). You must 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**.

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 **clsmuxpd**. This information is accessible by the routines in the APIs. Clinfo tracks topology events, as the cluster passes through various states. States that are tracked include:

- The cluster state is up or down
- The cluster substate has become stable or unstable
- A service adapter has failed and its IP address is being swapped to a standby adapter
- A service adapter on a node has completed the swap with its standby adapter
- A network has failed
- A node is in the process of joining the cluster
- A node has completed joining the cluster
- A node is leaving the cluster (that is, the node has failed)
- A node has left the cluster
- A new primary Cluster Manager has been elected (optional event)

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:

- The clusters
- The nodes in the clusters
- The network interfaces attached to each node

Cluster

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 (This information is available for applications that were developed with Version 2.1 of the Clinfo C API.)

Cluster Name

A cluster name uniquely identifies a cluster for a site. The cluster name is a string containing up to 31 characters.

Cluster ID

A cluster ID identifies each cluster in a network. The cluster ID is an arbitrary number ranging from 1 to 999,999,999. The cluster ID must be unique for a site.

Cluster State

A cluster can be in one of three defined states:

CLS_UP	At least one node in the cluster is up, and a primary is defined.
CLS_DOWN	At least one node in the cluster is up, but a primary is not yet defined.
CLS_UNKNOWN	**clsmuxpd** is unable to communicate, or is not yet communicating with an active **clstrmgr**.

Cluster Substate

A cluster can be in one of several defined substates:

CLSS_ERROR	A script has failed; the cluster has been in the process of configuration (unstable) for too long.

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 (a change in topology is occurring).
CLSS_UNKNOWN	**clsmuxpd** is unable to communicate with a **clstrmgr**.

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.)

Node

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
- 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 31 characters.

Node State

A node can be in one of four defined states:

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

Network Interfaces

The number and addresses of service interfaces attached to this node that have a state of CLS_UP. If this number goes to zero, the state of the node is changed to CLS_DOWN.

Network Interface

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

The service adapter is the primary connection between the node and the network. A node has one service adapter for each network to which it connects. The service adapter is used for all shared AIX network connections and is the primary interface for keepalive traffic from the **clstrmgr**. It is also the address published by Clinfo to application programs that want to use cluster services.

The total number of service adapters that a node has depends on the types and number of networks supported by the cluster. An HACMP cluster can support multiple networks and point-to-point connections, as well as an RS232 serial line point-to-point connection.

Available Network Interface Information

Clinfo maintains the following information about a network interface:

- Cluster ID
- Node name
- Interface name
- Interface ID
- Interface address
- Interface state

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.

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 numeric ID assigned to the interface by the Cluster Manager.

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 adapter or network is down.

CLS_INVALID This interface is not defined for this node.

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.

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.

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 the list above, you must specify the following include directive in each source module that uses the **cl_registereventnotify** routine:

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

In addition to the list above, you must specify the following include directive in each source module that uses the **cl_registerwithclsmuxpd** routine:

```
#include <cluster/clsnmp.h>
```

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  -lclstr
```

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

```
-lcl_r  -lclstr_r
```

The **libcl.a** and **libcl_r.a** libraries contain the routines that support the Clinfo C API. The **libclstr.a** and **libclstr_r.a** libraries hold commands and other information that may be used by some of the **libcl.a** or **libcl_r.a** routines.

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 (31). 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).

The constants CL_MAXCLUSTERS, CL_MAXNODES, and CL_MAXNETIFS, available in previous Clinfo releases, have been replaced by macros that provide the current sizes of the various arrays in the shared memory segment. 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.

The **clsmuxpd** API routine **cl_registerwithclsmuxpd** uses the following constant, defined in the **clsnmp.h** file:

CLSMUXPD_SVC_PORT The clsmuxpd-application deadman connection port as defined in **/etc/services**.

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

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
 };
```

Data Structure Representing a Network Interface

The following data structure represents a network interface:

```
struct cl_netif {
   int cli_interfaceid;          /* interface id */
   int cli_nodeid;               /* node id (internal use only)*/
   char cli_nodename[CL_MAXNAMELEN];/* node name */
   int cli_clusterid;            /* cluster id */
   enum cls_state cli_state;     /* interface state */
   char cli_name [CL_MAXNAMELEN];/* interface name */
   struct sockaddr_in cli_addr;  /* interface address */
};
```

Data Structure Representing a Node

The following data structure represents a cluster node:

```
struct cl_node {
   char cln_nodename[CL_MAXNAMELEN];/* node name */
   int cln_nodeid;       /* node id (internal use only */
   int cln_clusterid;    /* cluster id */
   enum cls_state cln_state;/* node state */
   int cln_nif;          /* number of interfaces */
   struct cl_netif *cln_if;/* interfaces */
};
```

Data Structure Representing a Cluster

The following data structure represents a cluster:

```
struct cl_cluster {
   int  clc_clusterid;   /* cluster id */
   enum cls_substate clc_substate;/* cluster substate */
   enum cls_state clc_state;              /* cluster state */
   char clc_primary[CL_MAXNAMELEN];    /* primary node name */
   char clc_name [CL_MAXNAMELEN];/* cluster name */
};
```

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 */
};
```

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  */
};
```

Data Structure Representing Application Registration Data

The **clsmuxpd** API routine **cl_registerwithclsmuxpd** uses the following data structure for application registration data, defined in the **clsnmp.h** file:

```
typedef struct _appData {
   int nodeId;            /* node id - internal use*/
   int  pid;              /* process id */
   char name[128];        / *application name */
   char version[128];     /* application version */
   char descr[128];       /* description of app */
} appData;
```

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 gives 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 example of the C API **cl_getprimary** routine, 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 example of the C API **cl_isaddravail** routine, 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 routines before and after the **cl_getclusters** and **cl_getnodemap** routines.

cl_alloc_clustermap Allocates storage for a list of four clusters of eight nodes with eight interfaces each.

cl_alloc_nodemap Allocates storage for a list of eight nodes with eight interfaces each.

cl_free_clustermap Frees the storage for a list of clusters that was allocated by calling **cl_alloc_clustermap**.

cl_free_nodemap Frees the storage for a list of clusters that was allocated by calling **cl_alloc_nodemap**.

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.

Previous example code:

```
int status;
struct cl_cluster clustermap[CL_MAXCLUSTERS];
status = cl_getclusters(clustermap);
if (status < 0) {
   cl_perror(status, "Can’t get cluster information");
} else { printf("There are currently %d running clusters",
      status);
}
```

New example code:

```
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);
```

Requests

The Clinfo C API has four types of requests:

- Cluster information requests
- Node information requests
- Network interface information requests
- Event notification registration 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.

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.

Application Data Registration Routine

The **clsmuxpd** API has one routine, **cl_registerwithclsmuxpd**. This routine registers an application with **clsmuxpd** on the same cluster node.

Utilities

The Clinfo C API has four utility routines:

cl_initialize	Verifies that Clinfo is running and sets up shared memory access.
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

Syntax

```
int cl_initialize ()
```

Description

The **cl_initialize** routine checks to see if Clinfo is running, and if so, acquires the shared memory map. This map is managed by **clinfo** using information stored in the MIB by **clsmuxpd**.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable errno for additional information.
CLE_IVSHMEM	No cluster information is available. This status usually indicates that the Clinfo daemon is not running.

Example

```
int status;
status = cl_initialize ();
if (status != CLE_OK) {
   cl_perror (status, "Unable to initialize clinfo");
} else {
   printf("Clinfo initialized.");
}
```

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_IVSHMEM, "Can’t service this request");
```

yields the output:

```
Can’t service this request: No cluster information is available.
```

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

```
int clusterid, status;
char  nodename[CL_MAXNAMELEN];
strcpy (nodename, "moby");
if ((status = cl_getlocalid(&clusterid, nodename)) < 0) {
   if (status == CLE_IVNODE) {
      printf("This node is not a cluster member");
   } else {
      cl_perror(status, "Can’t get local cluster ID");
   }
}
```

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, free the storage when you are done by calling the **cl_free_clustermap** routine.

Parameters

clustermap	A `**cl_cluster` pointer used as 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_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	A `**cl_node` pointer used as 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_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 do so.

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 first compares interfaces defined to HACMP for AIX as private (such as a serial optical channel) to local interfaces. If no match is found, the routine then compares 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 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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
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, status;
char nodename [CL_MAXNAMELEN];
struct sockaddr_in laddr, raddr;
clusterid = 1;
strcpy (nodename, "node1");
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", 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_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_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_IVSHMEM	No cluster information is available. This status usually indicates that the Clinfo daemon is not running.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

```
int clusterid, status;
struct cl_cluster clusterbuf;
/* get_state returns one of the values
 * listed in enum cls_state.
 */
clusterid = 1;
status = cl_getcluster(clusterid, &clusterbuf);
if (status != CLE_OK) {
   cl_perror(status, "Can't get cluster information");
} else {
   printf("Cluster %d is named: %s, State: %s",
      clusterid,
      clusterbuf.clc_name,get_state(clusterbuf.clc_state));
}
```

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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVCLUSTERNAME	The request specified an invalid cluster name.

Example

```
int clusterid;
char clustername[CL_MAXNAMELEN];

strcpy (clustername, "cluster_a");
clusterid = cl_getclusterid (clustername);
if (clusterid < 0) {
   cl_perror (clusterid, "Can't get cluster ID");
} else {
   printf ("ID of cluster %s is %d",
      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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
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

```
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 ("1.1.1.1");
clusterid = cl_getclusteridbyifaddr (&addr);
if (clusterid < 0) {
   cl_perror (clusterid,"Can’t get cluster ID");
} else {
   printf("ID of cluster with address %s is %d",
      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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
CLE_BADARGS	Missing or invalid parameters.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

```
int clusterid;
char interfacename[CL_MAXNAMELEN];
strcpy (interfacename, "if_moby");
clusterid = cl_getclusteridbyifname (interfacename);
if (clusterid < 0) {
   cl_perror (clusterid, "Can’t get cluster ID");
} else {
   printf ("ID of the cluster with interface %s is %d",
      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.

CLE_IVSHMEM Cannot access shared memory. Check to see if the Clinfo daemon is running.

Example

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

```
int i;
int numClusters;
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",
      cl_errmsg_r(numClusters, cbuf));
}
else
{
   printf("There are currently %d running clusters",
      numClusters);
   for(i=0; i < numClusters; i++)
   {
   printf("Cluster: %d (%s), state: %s, primary: %s",
      clustermap[i].clc_clusterid,  clustermap[i].clc_name,
      get_state(clustermap[i].clc_state), clustermap[i].clc_primary);
```

```
    }

   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 on the **cl_registereventnotify** routine page.

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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

```
int clusterid, status;
char interfacename[CL_MAXNAMELEN];
struct sockaddr_in addr;
strcpy (interfacename, "moby_en0");
clusterid = 1;
status = cl_getifaddr (clusterid, interfacename, &addr);
if (status != CLE_OK) {
   cl_perror (status, "Can't find interface address");
} else {
   printf ("Interface address for %s is %s",
      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 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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVADDRESS	The request specified an invalid network interface address.

Example

```
int clusterid, status;
struct sockaddr_in addr;
char interfacename[CL_MAXNAMELEN];
clusterid = 1;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr ("1.1.1.1");
status = cl_getifname (clusterid, &addr, interfacename);
if (status != CLE_OK) {
   cl_perror (status, "Can't find interface name");
}
else {
   printf ("Interface address for %s is %s",
      inet_ntoa (addr.sin_addr.s_addr), interfacename);
}
```

cl_getlocalid Routine

Syntax

```
int cl_getlocalid (int *clusteridp, 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

clusteridp	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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
CLE_IVNODE	Node is not a cluster node.

Example

```
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_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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.

Example

```
int clusterid,status;
char nodename[CL_MAXNAMELEN];
struct cl_node nodebuf;

clusterid = 1;
strcpy (nodename, "node1");
status = cl_getnode (clusterid, nodename, &nodebuf);
if (status != CLE_OK) {
   cl_perror(status, "Can't get node info");
} else {
   printf("
"Node %s on cluster %d state ", nodename, clusterid);
   switch (nodebuf.cln_state) {
      case CLS_UP:          printf("UP"); break;
      case CLS_DOWN:                printf("DOWN"); break;
      default:                      printf("???"); break;
   }
}
```

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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

```
int clusterid, status;
char interfacename[MAXNAMELEN];
struct sockaddr_in addr;
strcpy (interfacename, "editserver");
clusterid = 1;
status = cl_getnodeaddr(clusterid, interfacename, &addr);
if (status != CLE_OK) {
   cl_perror(status, "Can't get node addr");
} else {
   printf("Address of %s on cluster %d is %s",
      interfacename, clusterid, inet_ntoa(addr.sin_addr));
}
```

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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVADDRESS	The request specified an invalid network interface address.

Example

```
int clusterid, status;
char nodename [CL_MAXNAMELEN];
struct sockaddr_in addr;
clusterid = 1;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr ("1.1.1.1");
status = cl_getnodenamebyifaddr (clusterid, &addr, nodename);
if (status !=CLE_OK) {
   cl_perror(nodename,"Can't get node name");
} else {
   printf("Name of %s on cluster %d is %s",
      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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
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, 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_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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

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

```
int i, j, clusterid;
int nodes;
char cbuf[CL_ERRMSG_LEN];
struct cl_node *nodemap;
clusterid =1;
cl_alloc_nodemap (&nodemap);
if (nodemap==NULL){
   printf ("Storage problems/n");
}
nodes = cl_getnodemap(clusterid, nodemap);
if(nodes < 0)
    {
   printf("cl_getnodemap: (failed) %s",
      cl_errmsg_r(nodes, cbuf));
    }
printf("Cluster %d has %d nodes", clusterid,
   nodes);
```

```
for(i=0; i < nodes; i++)
    {
   printf("Node %s state: %s has interfaces: %d ",
      nodemap[i].cln_nodename,
      get_state(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_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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
CLE_NOPRIMARY	No information is available about the primary Cluster Manager. This status usually indicates that the user has not designated a primary Cluster Manager.
CLE_IVCLUSTER	The cluster is not available.

Example

```
int clusterid, status;
char nodename[CL_MAXNAMELEN];
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 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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
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, status;
char nodename[CL_MAXNAMELEN];
struct sockaddr_in addr;
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));
}
```

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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
CLE_IVCLUSTER	The specified cluster is not available.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

```
int clusterid, status;
clusterid = 1;
status = cl_isclusteravail (clusterid);
if (status != CLE_OK) {
   cl_perror (status, "Cluster not available");
} else {
   printf ("Cluster %d is available", 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_IVSHMEM	Cannot access shared memory. Check to see if the Clinfo daemon is running.
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, status;
char nodename[CL_MAXNAMELEN];
clusterid = 1;
strcpy (nodename, "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",
      nodename, clusterid);
}
```

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 which fits the specified description occurs. When this signal is received, the **cl_getevent** routine may be called to get information about the event which 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 later 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 which may be requested include the following:

CL_SWAPPING_ADAPTER—The specified adapter's interface is 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.

CL_SWAPPED_ADAPTER—The specified adapter interfaces 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.

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.

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.

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.

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.

CL_JOINING_NODE—The specified node is coming up. This event requires cluster and node identification.

CL_JOINED_NODE—The specified node is up. This event requires cluster and node identification.

CL_FAILING_NODE—The specified node is going down. This event requires cluster and node identification.

CL_FAILED_NODE—The specified node is down. This event requires cluster and node identification.

CL_NEW_PRIMARY—The specified cluster has a new primary node. This event requires cluster identification.

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_registerwithclsmuxpd Routine

Syntax

```
#include <cluster/clsnmp.h>
int cl_registerwithclsmuxpd(char *name, char *descr,
              char *version )
```

Description

The **cl_registerwithclsmuxpd** routine registers an application with **clsmuxpd**. Each registration request specifies an application name, description, and version number. (This routine handles the NodeId and PID fields of the appData structure.) If the routine is successful, a TCP-based “deadman” connection is created between the application and **clsmuxpd**. Users can make SNMP-based requests to **clsmuxpd** to determine which applications are currently registered. **clsmuxpd** sends out an SNMP trap whenever an application registers or dies.

Note: Only applications running on cluster nodes (nodes on which the **clstrmgr** daemon is running) may register with **clsmuxpd**. Applications running on clients cannot register with a **clsmuxpd** process.

The **cl_registerwithclsmuxpd** routine uses the following data structure for application registration data. The structure is defined in the **clsnmp.h** file:

```
typedef struct _appData {
   int nodeId;
   int pid;
   char name[128];
   char version[128];
   char descr[128];
} appData;
```

Parameters

name	The name of the application.
descr	A brief description of the application.
version	The version of the application.

Status Codes

CLE_OK	The request completed successfully.
CLE_SYSERR	System error. Check the AIX global variable errno for additional information.
CLE_NOSERVICE	The clsmuxpd port was not found in **/etc/services**. It is put there when you install the cluster.server component of HACMP for AIX. Make sure the application is running on a cluster node, not on a client.

Example

```
#include <cluster/clsnmp.h>
   .
   .
   .
   appData app;
   .
   .
   .
strcpy (app.name, argv[0]);
   strcpy (app.descr, "This is an example");
   strcpy (app.version, "1.0");
rc = cl_registerwithclsmuxpd (app.name, app.descr, app.version);
```

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 which 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

num_reqs The number of event notification unregistration requests being made (limit is 10).

enr_list The list of event notification unregistration requests. The events which 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.

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.

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.

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.

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.

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.

CL_JOINING_NODE—The specified node is coming up. This event requires cluster and node identification.

CL_JOINED_NODE—The specified node is up. This event requires cluster and node identification.

CL_FAILING_NODE—The specified node is going down. This event requires cluster and node identification.

CL_FAILED_NODE—The specified node is down. This event requires cluster and node identification.

CL_NEW_PRIMARY—The specified cluster has a new primary node. This event requires cluster identification. A value of -1 indicates all such components.

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.

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

See the example on the **cl_registereventnotify** page.

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 routines and utilities available in the Clinfo C++ API.

Before reading this chapter, you should read Chapter 1, Cluster Information Program, which describes the types of information about an HACMP for AIX cluster that the HACMP for AIX MIB contains. It may also be useful to read Chapter 2, Clinfo C API, which describes the C API. The C++ API routines call the C API routines.

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

Clinfo C++ Object Classes

The Clinfo C++ API has three object classes: clusters, nodes, and network interfaces. Each network interface belongs to a single node, and each node 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 as 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 which 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 three 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 its 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 which 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>
```

You must also specify the following include directive in each source module that uses the **cl_registereventnotify** Clinfo C API routine:

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

You must also specify the following include directives in each source module that uses the **cl_registerwithclsmuxpd** Cluster SMUX Peer API routine:

```
#include <cluster/clsnmp.h>
#include <signal.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. The **libclstr.a** and **libclstr_r.a** libraries hold commands and other functions that may be called by the Clinfo C library routines.

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

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 (31). 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).

The Cluster SMUX Peer API routine **cl_registerwithclsmuxpd** uses the following constant, defined in the **clsnmp.h** file:

CLSMUXPD_SVC_PORT The clsmuxpd-application deadman connection port as defined in **/etc/services**.

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 shared memory segment. 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.

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;
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;
};
```

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
   int CL_getallinfo(CL_node*, 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.

Functions 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 three classes, **CL_cluster**, **CL_netif**, and **CL_node**, are overloaded in order to assign the C structures **cl_cluster**, **cl_netif**, 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

```
int cl_initialize();
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 gives 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 three types of requests:

- Cluster information requests
- Node information requests
- Network interface information requests

Cluster Information Requests

The following cluster information requests return information about a cluster:

CL_getallinfo	Calls the C function **cl_getnodemap.** 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.

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 which 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 functions **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 functions **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.

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.

Application Data Registration Requests

The application data registration routine **cl_registerwithclsmuxpd** registers an application with **clsmuxpd** on the same cluster node. You must call this routine from the **clsmuxpd** C API.

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.
CLE_IVSHMEM	Cannot access shared memory. Check to see if Clinfo daemon is running.

Example

```
{
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);
}
```

CL_getlocalid Routine

Syntax

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

Description

Returns a **CL_node** object which 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_IVSHMEM	Cannot access shared memory. Check to see if Clinfo daemon is running.
CLE_IVNODE	Node is not a cluster node.

Example

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

```
// This function is not a member of a class.
{
CL_status s;
CL_node lnode;
char cbuf[CL_ERRMSG_LEN];
   node = 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_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_IVSHMEM	Cannot access shared memory. Check to see if Clinfo daemon is running.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.

Example

```
{
CL_cluster clus;
CL_status s;
CL_node nodes[8];
CL_node *ret = &nodes[0];
int numnodes;
clus.clc_clusterid = 2;
   numnodes = clus.CL_getallinfo(ret, s);
   if (s < 0)
      cl_perror(s, progname);
   printf("number of nodes in cluster: %d", numnodes);
   for (int i=0; i<numnodes; i++) {
      printf("[%d] cl %d", i, ret[i].cln_clusterid);
      printf("no %s", ret[i].cln_nodename);
      printf("st %d", ret[i].cln_state);
      printf("ni %d", 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_IVSHMEM Cannot access shared memory. Check to see if Clinfo daemon is running.

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

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

```
{
CL_cluster clus;
CL_status s;
CL_clusterid clid;
char cbuf[CL_ERRMSG_LEN];
strcpy(clus.clc_name.name, "cluster_2");
   clid = clus.CL_getclusterid(s);
if (s < 0)
   cl_errmsg_r(s, cbuf);
printf("clusterid = %d", clid);
}
```

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_IVSHMEM Cannot access shared memory. Check to see if Clinfo daemon is running.

CLE_IVCLUSTERID The request specified an invalid cluster ID.

Example

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

```
{
CL_cluster clus;
CL_status s;
CL_cluster ret;
char cbuf[CL_ERRMSG_LEN];
clc_clusterid = 2;
   ret = clus.CL_getinfo(s);
if (s < 0)
   cl_errmsg_r(s, cbuf);
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", 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 whose 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_IVSHMEM Cannot access shared memory. Verify Clinfo daemon is running.

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

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

```
{
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);
}
```

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_IVSHMEM Cannot access shared memory. Check to see if Clinfo daemon is running.

CLE_IVCLUSTER The request specified an invalid cluster.

CLE_IVCLUSTERID The request specified an invalid cluster ID.

Example

```
{
CL_cluster clus;
CL_status s;
clus.clusterid = 2;
   s = clus.CL_isavail();

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

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. Or, 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_IVSHMEM Cannot access shared memory. Check to see if Clinfo daemon is running.

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.

Multi-threaded Example

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

```
// CL_netif::CL_getclusterid by ifaddr
{
CL_status s;
char cbuf[CL_ERRMSG_LEN];
CL_clusterid clid;
CL_netif netif;
char *addr = "2.3.4.5";
```

```
netif.cli_addr.sin_family = AF_INET;
netif.cli_addr.sin_addr.s_addr = inet_addr(addr);
netif.cli_name.name[0] = NULL;
   clid = netif.CL_getclusterid(s);
if (s < 0)
   cl_errmsg_r(s, cbuf);
printf("clusterid = %d", clid);
}
```

Single-threaded 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_netif::CL_getclusterid by ifname
{
CL_status s;
CL_clusterid clid;
CL_netif netif;
strcpy(netif.cli_name.name, "moby_en2_cl2");
netif.cli_addr.sin_addr.s_addr = NULL;
   clid = netif.CL_getclusterid(s);
if (s < 0)
   cl_errmsg(s);
printf("clusterid = %d", clid);
}
```

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_IVSHMEM	Cannot access shared memory. Check to see if Clinfo daemon is running.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

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

```
{
CL_ifaddr ifaddr;
CL_netif netif;
char cbuf[CL_ERRMSG_LEN];
netif.cli_clusterid = 2;
```

```
strcpy(netif.cli_name.name, "moby_en2_cl2");
   ifaddr = netif.CL_getifaddr(s);
if (s < 0)
   cl_errmsg_r(s, cbuf);
printf("ifaddr = %s", 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_IVSHMEM	Cannot access shared memory. Check to see if Clinfo daemon is running.
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

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

```
// CL_netif::CL_getifname get interfacename given clusterid and nodename
{
CL_status s;
char cbuf[CL_ERRMSG_LEN];
CL_ifname ifname;
CL_netif netif;
netif.cli_addr.sin_addr.s_addr = NULL;
netif.cli_clusterid = 2;
strcpy (netif.cli_nodename.name, "moby");
   ifname = netif.CL_getifname(s);
if (s < 0)
   cl_errmsg_r(s, cbuf);
printf("ifname = %s", 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_IVSHMEM	Cannot access shared memory. Check to see if Clinfo daemon is running.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNETIFNAME	The request specified an invalid network interface name.

Example

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

```
{
CL_status s;
CL_netif netif;
CL_ifaddr ifaddr;
char cbuf[CL_ERRMSG_LEN];
netif.cli_clusterid = 2;
```

```
strcpy(netif.cli_name.name, "moby_en2_cl2");
   ifaddr = netif.CL_getnodeaddr(s);
if (s < 0)
   cl_errmsg_r(s, cbuf);
printf("ifaddr = %s", 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 given a cluster ID and a network interface address.

OR

Returns the node name given 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_IVSHMEM	Cannot access shared memory. Check to see if Clinfo daemon is running.
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

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

```
{
CL_status s;
char cbuf[CL_ERRMSG_LEN];
CL_nodename nname;
CL_netif netif;
char *addr = "2.3.4.5";
netif.cli_clusterid = 2;
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(s);
if (s < 0)
   cl_errmsg_r(s, cbuf);
printf("node name = %s", nname.name);
}
{
CL_status s;
char cbuf[CL_ERRMSG_LEN];
CL_nodename nname;
CL_netif netif;
netif.cli_clusterid = 2;
strcpy(netif.cli_name.name, "moby_en2_cl2");
netif.cli_addr.sin_addr.s_addr = NULL;
   nname = netif.CL_getnodenamebyif(s);

if (s < 0)
   cl_errmsg_r(s, cbuf);
printf("node name = %s", 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_IVSHMEM Cannot access shared memory. Check to see if Clinfo daemon is running.

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 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_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_IVSHMEM	Cannot access shared memory. Check to see if Clinfo daemon is running.
CLE_NOROUTE	No route available.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.

Example

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

```
{
CL_status s;
CL_node node;
CL_route route;
char cbuf[CL_ERRMSG_LEN];
node.cln_clusterid = 2;
strcpy(node.cln_name.name, "moby");
   route = node.CL_bestroute(s);
if (s < 0)
   cl_errmsg_r(s, cbuf);
// don't call inet_ntoa twice in one printf!
printf("local = %s",
   inet_ntoa(route.localaddr.sin_addr));
printf("remote = %s",
   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 which 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_IVSHMEM	Cannot access shared memory. Check to see if Clinfo daemon is running.
CLE_IVCLUSTERID	The request specified an invalid cluster ID.
CLE_IVNODENAME	The request specified an invalid node name.

Example

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

```
{
CL_status s;
CL_node node;
CL_node ret;
char cbuf[CL_ERRMSG_LEN];
node.cln_clusterid = 2;
strcpy(node.cln_name.name, "moby");
```

```
   ret = node.CL_getinfo(s);
if (s < 0)
   cl_errmsg_r(s, cbuf);
printf("clusterid %d", ret.cln_clusterid);
printf("nodename %s", ret.cln_nodename.name);
printf("state %d", ret.cln_state);
printf("nif %d", 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_IVSHMEM Cannot access shared memory. Check to see if Clinfo daemon is running.

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 s;
node.cli_clusterid = 2;
strcpy(node.cln_name.name, "moby");
   s = node.CL_isavail();

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

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 adapter 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,
          # clearing of lockfiles, 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_IVSHMEM:  /* no cluster information available */
                 cl_perror(node_status, "node unavailable");
                 exit(2);
                 break;

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

Appendix A HACMP for AIX MIB

This appendix identifies the objects and traps defined in the HACMP for AIX MIB, lists the HACMP for AIX SNMP MIB definition, and briefly describes the available AIX SNMP commands and functions you can use to query a MIB. See the AIX documentation for more detailed information.

HACMP for AIX Objects Defined in the MIB

The HACMP for AIX MIB maintains information about the following objects:

Cluster	Describes the attributes of a cluster.
Node	Describes the attributes of the nodes within a cluster.
Address	Describes the attributes of the network addresses of cluster nodes.
Network	Describes the attributes of the networks that support the cluster.
Cluster Manager	Maintains information on the **clstrmgr** daemon.
Cluster Lock Manager	Maintains information on the **cllockd** daemon.
Cluster Information	Maintains information on the **clinfo** daemon.
Application	Maintains information about the applications registered with the **clsmuxpd** daemon.
Cluster SMUX Peer	Maintains statistical information about the **clsmuxpd** daemon.
Event	Maintains information on the set of 1000 most recent cluster events.
Resmanager	Maintains information on the **clresmgrd** daemon (HACMP/ES only.)

HACMP for AIX SNMP MIB Definition

The SNMP MIB provided with the HACMP for AIX software is listed below.

```
\-- I. HEADER
--
--A) High Availability Cluster Multi-Processing for AIX Cluster
--         SNMP Peer MIB Definition.
--
 RISC6000CLSMUXPD-MIB{ 1 3 6 1 4 1 }
--
 DEFINITIONS ::= BEGIN
--
--B) Imported syntaxes.
--
 IMPORTS
         enterprises, IpAddress, Counter, OBJECT-TYPE
                 FROM RFC1065-SMI
         DisplayString
                 FROM RFC1213-MIB;
--
--C) The root of the RISC6000CLSMUXPD-MIB is as follows:
--
        ibm               OBJECT IDENTIFIER    ::= { enterprises 2 }
        ibmAgents         OBJECT IDENTIFIER    ::= { ibm 3 }
        aix               OBJECT IDENTIFIER    ::= { ibmAgents 1  }
        aixRISC6000       OBJECT IDENTIFIER    ::= { aix 2 }
        risc6000agents    OBJECT IDENTIFIER    ::= { aixRISC6000 1 }
        risc6000clsmuxpd  OBJECT IDENTIFIER    ::= { risc6000agents 5 }
--
        cluster           OBJECT IDENTIFIER    ::= { risc6000clsmuxpd 1 }
        node              OBJECT IDENTIFIER    ::= { risc6000clsmuxpd 2 }
        address           OBJECT IDENTIFIER    ::= { risc6000clsmuxpd 3 }
        network           OBJECT IDENTIFIER    ::= { risc6000clsmuxpd 4 }
--
        clstrmgr          OBJECT IDENTIFIER    ::= { risc6000clsmuxpd 5 }
        cllockd           OBJECT IDENTIFIER    ::= { risc6000clsmuxpd 6 }
        clinfo            OBJECT IDENTIFIER    ::= { risc6000clsmuxpd 7 }
--
        application       OBJECT IDENTIFIER    ::= { risc6000clsmuxpd 8 }
--
        clsmuxpd          OBJECT IDENTIFIER    ::= { risc6000clsmuxpd 9 }
        event             OBJECT IDENTIFIER    ::= { risc6000clsmuxpd 10 }
--
        resmanager        OBJECT IDENTIFIER    ::= { risc6000clsmuxpd 11 }
--
--
-- II. The Cluster Group
--
--A) clusterId
--        This field is read from the HACMP for AIX object repository.
--
 clusterIdOBJECT-TYPE
         SYNTAX  INTEGER
         ACCESS  read-only
         STATUS  mandatory
         DESCRIPTION
                 "The ID of the cluster"
         ::= { cluster 1 }
--
--B) clusterName
--        This field is read from the HACMP for AIX object repository.
--
```

```
  clusterNameOBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “User configurable cluster Name”
          ::= { cluster 2 }
--
--C) clusterConfiguration
--        This field is read from the HACMP for AIX object repository.
--
  clusterConfigurationOBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  deprecated
          DESCRIPTION
                  “The cluster configuration”
          ::= { cluster 3 }
--
--D) clusterState
--        This field is returned by the clstrmgr.
--
  clusterStateOBJECT-TYPE
          SYNTAX  INTEGER { up(2), down(4),
                            unknown(8) }
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The cluster status”
          ::= { cluster 4 }

  trapClusterStateTRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ clusterState, clusterId, clusterNodeId }
          DESCRIPTION
                  “Fires whenever the cluster changes state.”
          ::= 10

--
--E) clusterPrimary
--        This field is returned by the clstrmgr.
--
--
  clusterPrimaryOBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The Node ID of the Primary Lock Manager”
          ::= { cluster 5 }

--
--F) clusterLastChange
--        This field is a integer string returned by the gettimeofday()
--        library call and is updated if any cluster, node,
--        or address information changes.
--
  clusterLastChangeOBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “Time in seconds of last change in this cluster.”
          ::= { cluster 6 }
--
```

```
--G) clusterGmtOffset
--        This field is a integer string returned by the gettimeofday()
--        library call and is updated if any cluster, node,
--        or address information changes.
--
  clusterGmtOffsetOBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “Seconds west of GMT for the time of last change in this cluster.”
          ::= { cluster 7 }
--
--H) clusterSubState
--        This field is returned by the clstrmgr.
--
--
  clusterSubStateOBJECT-TYPE
          SYNTAX  INTEGER { unstable(16), error(64),
                            stable(32), unknown(8), reconfig(128) }
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The cluster substate”
          ::= { cluster 8 }

  trapClusterSubStateTRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ clusterSubState, clusterId, clusterNodeId }
          DESCRIPTION
                  “Fires whenever the cluster changes substate.”
          ::= 11

--
--I) clusterNodeName
--        This field is read from the HACMP for AIX object repository.
--
  clusterNodeNameOBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “User configurable cluster local node name”
          ::= { cluster 9 }
--
--      J) clusterPrimaryNodeName
--              This field is returned by the clstrmgr.
--
--
  clusterPrimaryNodeName  OBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The Node Name of the primary cluster node”
          ::= { cluster 10 }

  trapNewPrimaryTRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ clusterPrimary, clusterId, clusterNodeId }
          DESCRIPTION
                  “Fires whenever the primary node changes.”
          ::= 15
--
--      K) clusterNumNodes
```

```
--              This field is returned by the clstrmgr.
--
--
 clusterNumNodes  OBJECT-TYPE
         SYNTAX  INTEGER
         ACCESS  read-only
         STATUS  mandatory
         DESCRIPTION
                 “The number of nodes in the cluster”
         ::= { cluster 11 }
--
--L) clusterNodeId
--        This field is read from the HACMP for AIX object repository.
--
 clusterNodeIdOBJECT-TYPE
         SYNTAX  INTEGER
         ACCESS  read-only
         STATUS  mandatory
         DESCRIPTION
                 “The ID of the local node”
         ::= { cluster 12 }
--
--
-- III. The node group
--
--A) The node table
--        This is a variable length table which is indexed by
--        the node Id.
--
 nodeTableOBJECT-TYPE
         SYNTAX  SEQUENCE OF NodeEntry
         ACCESS  not-accessible
         STATUS  mandatory
         DESCRIPTION
                 “A series of Node descriptions”
         ::= { node 1 }
--
 nodeEntryOBJECT-TYPE
         SYNTAX  NodeEntry
         ACCESS  not-accessible
         STATUS  mandatory
         INDEX   { nodeId }
         ::= { nodeTable 1 }
--
 NodeEntry::= SEQUENCE {
         nodeId          INTEGER,
         nodeStateINTEGER,
         nodeNumIfINTEGER,
         nodeNameDisplayString
 }
--
--B) nodeId
--        This field is read from the HACMP for AIX object repository.
--
 nodeId  OBJECT-TYPE
         SYNTAX  INTEGER
         ACCESS  read-only
         STATUS  mandatory
         DESCRIPTION
                 “The ID of the Node”
         ::= { nodeEntry 1 }
--
--C) nodeState
--        This row is returned by the clstrmgr.
--
```

```
--
  nodeStateOBJECT-TYPE
          SYNTAX  INTEGER { up(2), down(4),
                            joining(32), leaving(64) }
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The State of the Node”
          ::= { nodeEntry 2 }

  trapNodeStateTRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ nodeState, clusterId, clusterNodeId }
          DESCRIPTION
                  “Fires whenever a node changes state.”
          ::= 12
--
--D) nodeNumIf
--        This row is returned by the clstrmgr.
--
--
  nodeNumIfOBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The number of network interfaces in this node”
          ::= { nodeEntry 3 }
--
--E) nodeName
--        This row is returned by the clstrmgr.
--
--
  nodeNameOBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The name of this node”
          ::= { nodeEntry 4 }
--
--
-- IV. The address group
--
--A) The address table
--        This is a variable length table which is indexed by
--        the node Id and the dotted decimal IP address.
--
  addrTableOBJECT-TYPE
          SYNTAX  SEQUENCE OF AddrEntry
          ACCESS  not-accessible
          STATUS  mandatory
          DESCRIPTION
                  “A series of IP address descriptions”
          ::= { address 1 }
--
  addrEntryOBJECT-TYPE
          SYNTAX  AddrEntry
          ACCESS  not-accessible
          STATUS  mandatory
          INDEX   { addrNodeId, addrAddress }
          ::= { addrTable 1 }
--
  AddrEntry::= SEQUENCE {
          addrNodeId  INTEGER,
```

```
        addrAddress  IpAddress,
        addrLabel  DisplayString,
        addrRole  INTEGER,
        addrNetId  INTEGER,
        addrState  INTEGER,
        addrActiveNode  INTEGER,
        oldAddrActiveNode INTEGER
 }
--
--B) addrNodeId
--      This field is read from the HACMP for AIX object repository.
--
 addrNodeIdOBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The ID of the Node this IP address is configured"
        ::= { addrEntry 1 }
--
--C) addrAddress
--      This field is read from the HACMP for AIX object repository.
--
 addrAddressOBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The IP address"
        ::= { addrEntry 2 }
--
--D) addrLabel
--      This field is read from the HACMP for AIX object repository.
--
 addrLabelOBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The IP label associated with the IP address"
        ::= { addrEntry 3 }
--
--D) addrRole
--      This field is read from the HACMP for AIX object repository.
--
 addrRoleOBJECT-TYPE
        SYNTAX  INTEGER { service(16), sharedService(128),
                          standby(32), boot(64) }
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The role of the IP address"
        ::= { addrEntry 4 }
--
--E) addrNetId
--      This field is read from the HACMP for AIX object repository.
--      It is provide so that clients can determine the corresponding
--      index into the network table.
--
 addrNetIdOBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The network ID of the IP address"
```

```
        ::= { addrEntry 5 }
--
--F) addrState
--      This field is returned from the Cluster Manager.
--
 addrStateOBJECT-TYPE
        SYNTAX  INTEGER { up(2), down(4), unknown(8) }
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The state of the IP address"
        ::= { addrEntry 6 }

 trapAddressStateTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ addrState, addrNetId, clusterId, clusterNodeId }
        DESCRIPTION
                "Fires whenever a address changes state."
        ::= 14

 trapAdapterSwapTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ addrState, clusterId, clusterNodeId }
        DESCRIPTION
                "Fires whenever a address swap occurs."
        ::= 17

--
--G) addrActiveNode
--      This field is returned from the Cluster Manager.
--
 addrActiveNodeOBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The ID of the Node on which this IP address is active"
        ::= { addrEntry 7 }
--
--      H) oldAddrActiveNode
--              This field is returned from the Cluster Manager.
--
 oldAddrActiveNode OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS not-accessible
        STATUS  mandatory
        DESCRIPTION
                "The ID of the Node on which this IP address was previously
                active"
        ::= { addrEntry 8 }

 trapAddressTakeover  TRAP-TYPE
        ENTERPRISE   risc6000clsmuxpd
        VARIABLES    { addrActiveNode, oldAddrActiveNode,
                               clusterId, clusterNodeId }
        DESCRIPTION
                           "Fires whenever IP address takeover occurs."
        ::= 19
--
--
-- V. The network group
--
--A) The network table
--      This is a variable length table index by node Id
--      and network Id.
```

```
--
 netTable        OBJECT-TYPE
         SYNTAX  SEQUENCE OF NetEntry
         ACCESS  not-accessible
         STATUS  mandatory
         DESCRIPTION
                 “A series of Network descriptions”
         ::= { network 1 }
--
 netEntry        OBJECT-TYPE
         SYNTAX  NetEntry
         ACCESS  not-accessible
         STATUS  mandatory
         INDEX   { netNodeId, netId }
         ::= { netTable 1 }
--
 NetEntry        ::= SEQUENCE {
         netNodeIdINTEGER,
         netId           INTEGER,
         netName         DisplayString,
         netAttributeINTEGER,
         netStateINTEGER
 }
--
--B) netNodeId
--        This field is read from the HACMP for AIX object repository.
--
 netNodeIdOBJECT-TYPE
         SYNTAX  INTEGER
         ACCESS  read-only
         STATUS  mandatory
         DESCRIPTION
                 “The ID of the Node this network is configured”
         ::= { netEntry 1 }
--
--C) netId
--        This field is read from the HACMP for AIX object repository.
--
 netId   OBJECT-TYPE
         SYNTAX  INTEGER
         ACCESS  read-only
         STATUS  mandatory
         DESCRIPTION
                 “The ID of the network”
         ::= { netEntry 2 }
--
--D) netName
--        This field is read from the HACMP for AIX object repository.
--
 netName OBJECT-TYPE
         SYNTAX  DisplayString
         ACCESS  read-only
         STATUS  mandatory
         DESCRIPTION
                 “The name of network”
         ::= { netEntry 3 }
--
--E) netAttribute
--        This field is read from the HACMP for AIX object repository.
--
 netAttributeOBJECT-TYPE
         SYNTAX  INTEGER { public(2), private(1), serial(4) }
         ACCESS  read-only
         STATUS  mandatory
         DESCRIPTION
```

```
                “The attribute of the network.”
        ::= { netEntry 4 }
--
--F) netState
--      This row is returned by the clstrmgr.
--
--
 netStateOBJECT-TYPE
        SYNTAX  INTEGER { up(2), down(4), joining(32), leaving(64) }
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                “The state of the network”
        ::= { netEntry 5 }

 trapNetworkStateTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ netState, clusterId, clusterNodeId }
        DESCRIPTION
                “Fires whenever a network changes state.”
        ::= 13
--
--
-- VI. The Cluster Manager (clstrmgr) group
--
--A) The clstrmgr table
--      This is a variable length table which is indexed by
--      the node Id.
--
 clstrmgrTableOBJECT-TYPE
        SYNTAX  SEQUENCE OF ClstrmgrEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                “A series of clstrmgr process entries”
        ::= { clstrmgr 1 }
--
 clstrmgrEntryOBJECT-TYPE
        SYNTAX  ClstrmgrEntry
        ACCESS  not-accessible
        STATUS  mandatory
        INDEX   { clstrmgrNodeId }
        ::= { clstrmgrTable 1 }
--
 ClstrmgrEntry::= SEQUENCE {
        clstrmgrNodeIdINTEGER,
        clstrmgrVersionDisplayString,
        clstrmgrStatusINTEGER
 }
--
--B) clstrmgrNodeId
--      This field is determined by the IP address used to connect
--      to the clstrmgr.
--
 clstrmgrNodeIdOBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                “The node ID of the Cluster Manager”
        ::= { clstrmgrEntry 1 }
--
--C) clstrmgrVersion
--      This field is returned by the srcstat() library call.
--
```

```
  clstrmgrVersionOBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The version of the Cluster Manager”
          ::= { clstrmgrEntry 2 }
--
--D) clstrmgrStatus
--        This field is returned by the srcstat() library call or
--        set by a management station.
--
  clstrmgrStatusOBJECT-TYPE
          SYNTAX  INTEGER { up(2), down(4), suspended(16), unknown(8),
                            graceful(32), forced(64), takeover(128) }
          ACCESS  read-write
          STATUS  mandatory
          DESCRIPTION
                  “The state of the Cluster Manager”
          ::= { clstrmgrEntry 3 }
--
--
-- VII. The Cluster Lock Daemon (cllockd) group
--
--A) The cllockd table
--        This is a variable length table which is indexed by
--        the node Id.
--
  cllockdTableOBJECT-TYPE
          SYNTAX  SEQUENCE OF CllockdEntry
          ACCESS  not-accessible
          STATUS  mandatory
          DESCRIPTION
                  “A series of cllockd process entries”
          ::= { cllockd 1 }
--
  cllockdEntryOBJECT-TYPE
          SYNTAX  CllockdEntry
          ACCESS  not-accessible
          STATUS  mandatory
          INDEX   { cllockdNodeId }
          ::= { cllockdTable 1 }
--
  CllockdEntry::= SEQUENCE {
          cllockdNodeIdINTEGER,
          cllockdVersionDisplayString,
          cllockdStatusINTEGER
  }
--
--B) cllockdNodeId
--        This field is determined by the IP address used to connect
--        to the cllockd.
--
  cllockdNodeIdOBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The node ID of the Lock Manager”
          ::= { cllockdEntry 1 }
--
--C) cllockdVersion
--        This field is returned by the srcstat() library call.
--
  cllockdVersionOBJECT-TYPE
```

```
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  "The version of the Lock Manager"
          ::= { cllockdEntry 2 }
--
--D) cllockdStatus
--        This field is returned by the srcstat() library call or
--        set by a management station.
--
  cllockdStatusOBJECT-TYPE
          SYNTAX  INTEGER { up(2), down(4), unknown(8),
                                    suspended(16), stalled(256) }
          ACCESS  read-write
          STATUS  mandatory
          DESCRIPTION
                  "The state of the Lock Manager"
          ::= { cllockdEntry 3 }
--
--
-- VIII. The Client Information Daemon (clinfo) group
--
--A) The clinfo table
--        This is a variable length table which is indexed by
--        the node Id.
--
  clinfoTableOBJECT-TYPE
          SYNTAX  SEQUENCE OF ClinfoEntry
          ACCESS  not-accessible
          STATUS  mandatory
          DESCRIPTION
                  "A series of clinfo process entries"
          ::= { clinfo 1 }
--
  clinfoEntryOBJECT-TYPE
          SYNTAX  ClinfoEntry
          ACCESS  not-accessible
          STATUS  mandatory
          INDEX   { clinfoNodeId }
          ::= { clinfoTable 1 }
--
  ClinfoEntry::= SEQUENCE {
          clinfoNodeIdINTEGER,
          clinfoVersionDisplayString,
          clinfoStatusINTEGER
}
--
--B) clinfoNodeId
--        This field is determined by the IP address used to connect
--              to clinfo.
  clinfoNodeIdOBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  "The node ID of the Client Information Daemon"
          ::= { clinfoEntry 1 }
--
--C) clinfoVersion
--        This field is returned by the srcstat() library call.
--
  clinfoVersionOBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
```

```
        STATUS  mandatory
        DESCRIPTION
                "The version of the Client Information Daemon"
        ::= { clinfoEntry 2 }
--
--D) clinfoStatus
--      This field is returned by the srcstat() library call or
--      set by a management station.
--
 clinfoStatusOBJECT-TYPE
        SYNTAX  INTEGER { up(2), down(4), unknown(8), suspended(16) }
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "The state of the Client Information Daemon"
        ::= { clinfoEntry 3 }
--
--
-- IX. The Application Group
--
--A) The application table
--      This is a variable length table which is indexed by
--      the node Id followed by the process Id.
--
--      A library call will be provided which will allow
--      applications to register with the HACMP for AIX-SMUX peer.
--
 appTableOBJECT-TYPE
        SYNTAX  SEQUENCE OF AppEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "A series of application entries"
        ::= { application 1 }
--
 appEntryOBJECT-TYPE
        SYNTAX  AppEntry
        ACCESS  not-accessible
        STATUS  mandatory
        INDEX   { appNodeId, appPid }
        ::= { appTable 1 }
--
 AppEntry::= SEQUENCE {
        appNodeIdINTEGER,
        appPid          INTEGER,
        appName         DisplayString,
        appVersionDisplayString,
        appDescrDisplayString
 }
--
--B) appNodeId
--      This field is passed to the HACMP for AIX-SMUX peer.
--
 appNodeIdOBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The node ID of the application"
        ::= { appEntry 1 }
--
--C) appPid
--      This field is passed to the HACMP for AIX-SMUX peer.
--
 appPid  OBJECT-TYPE
```

```
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  "The process ID of the application"
          ::= { appEntry 2 }
--
--D) appName
--        This field is passed to the HACMP for AIX-SMUX peer.
--
  appName OBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  "The name of the application"
          ::= { appEntry 3 }

  trapAppStateTRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ appName, clusterId, clusterNodeId }
          DESCRIPTION
                  "Fires whenever an application is added or deleted."
          ::= 16

--
--E) appVersion
--        This field is passed to the HACMP for AIX-SMUX peer.
--
  appVersionOBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  "The version of the application"
          ::= { appEntry 4 }
--
--F) appDescr
--        This field is passed to the HACMP for AIX-SMUX peer.
--
  appDescrOBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  "The description of the application"
          ::= { appEntry 5 }
--
--
-- X. The Resource Group
--
--A) The Resource Group Table
--
  resGroupTableOBJECT-TYPE
          SYNTAX  SEQUENCE OF ResGroupEntry
          ACCESS  not-accessible
          STATUS  mandatory
          DESCRIPTION
                  "A series of Resource Group descriptions"
          ::= { resmanager 1 }
--
  resGroupEntryOBJECT-TYPE
          SYNTAX  ResGroupEntry
          ACCESS  not-accessible
          STATUS  mandatory
```

```
          DESCRIPTION
                  “Individual Resource Group description”
          INDEX           { resGroupId }
          ::= { resGroupTable 1 }
--
  ResGroupEntry   ::= SEQUENCE {
          resGroupId      INTEGER,
          resGroupName    DisplayString,
          resGroupPolicy  INTEGER,
          resGroupUserPolicyNameDisplayString,
          resGroupNumResourcesINTEGER,
          resGroupNumNodesINTEGER
  }
--
--B) Resource Group Id
--
--
  resGroupIdOBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The ID of the Resource Group”
          ::= { resGroupEntry 1 }

  trapRGAddTRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ resGroupId }
          DESCRIPTION
                  “Fires whenever a resource group is added.”
          ::= 20

  trapRGDelTRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ resGroupId }
          DESCRIPTION
                  “Fires whenever a resource group is deleted.”
          ::= 21

--
--C) Resource Group Name
--
--
  resGroupNameOBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The name of the Resource Group”
          ::= { resGroupEntry 2 }

--
--D) Resource Group Policy
--
--
  resGroupPolicyOBJECT-TYPE
          SYNTAX  INTEGER {
                          cascading(1),
                          rotating(2),
                          concurrent(3),
                          userdefined(4)
                          }
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
```

```
                “The State of the Resource Group”
        ::= { resGroupEntry 3 }

--
--E) Resource Group User-Defined Policy Name
--
--
  resGroupUserPolicyNameOBJECT-TYPE
        SYNTAX          DisplayString
        ACCESS          read-only
        STATUS          mandatory
        DESCRIPTION
                “The name of the user-defined policy”
        ::= { resGroupEntry 4 }

--
--F) Number of Resources in a Resource Group
--
--
  resGroupNumResourcesOBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                “The number of resources defined in the group”
        ::= { resGroupEntry 5 }

--
--G) Number of Participating Nodes in a Resource Group
--
--
  resGroupNumNodesOBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                “The number of participating nodes in the group”
        ::= { resGroupEntry 6 }

  trapRGChange    TRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ resGroupId, resGroupPolicy,
                resGroupNumResources, resGroupNumNodes }
        DESCRIPTION
                “Fires whenever the policy, number of nodes,
                or the number of resources of a resource
                group is changed.”
        ::= 22

--
--
-- XI. The Resources
--
--A) The Resource Table
--
--
  resTableOBJECT-TYPE
        SYNTAX  SEQUENCE OF ResEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                “A series of Resource descriptions”
        ::= { resmanager 2 }
--
  resEntryOBJECT-TYPE
```

```
          SYNTAX  ResEntry
          ACCESS  not-accessible
          STATUS  mandatory
          DESCRIPTION
                  “Individual Resource descriptions”
          INDEX   { resGroupId, resourceId }
          ::= { resTable 1 }
--
  ResEntry        ::= SEQUENCE {
          resourceGroupIdINTEGER,
          resourceIdINTEGER,
          resourceNameDisplayString,
          resourceTypeINTEGER
  }
--
--B) The Resource Group Id
--
--
        resourceGroupId OBJECT-TYPE
                SYNTAXINTEGER
                ACCESSread-only
                STATUSmandatory
                DESCRIPTION
                        “The ID of the resource group”
                ::= { resEntry 1 }

--
--C) Resource Id
--
--
  resourceIdOBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The ID of the Resource”
          ::= { resEntry 2 }

--
--D) Resource Name
--
--
  resourceNameOBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The name of this resource”
          ::= { resEntry 3 }

--
--E) Resource Type
--
--
-
    resourceType OBJECT-TYPE
        SYNTAX      INTEGER {
                    serviceLabel(1000), iPLabel(1000),
                    htyServiceLabel(1001),
                    fileSystem(1002),
                    volumeGroup(1003),
                    disk(1004), rawDiskPVID(1004),
                    aixConnectionServices(1005),
                    application(1006),
                    concurrentVolumeGroup(1007),
```

```
                    haCommunicationLinks(1008)
                }

          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The Type of the Resource”
          ::= { resEntry 4 }
--
-- XII. The Resource Group Node State
--
--A) The Resource Group Node State Table
--              The participating nodes and the current location of a given resource
--              group are determined and maintained via this table and indexed by
--              resource group ID and node ID.
--
--
  resGroupNodeTableOBJECT-TYPE
          SYNTAX  SEQUENCE OF ResGroupNodeEntry
          ACCESS  not-accessible
          STATUS  mandatory
          DESCRIPTION
                  “A series of resource group and associated node state descriptions”
          ::= { resmanager 3 }
--
  resGroupNodeEntryOBJECT-TYPE
          SYNTAX  ResGroupNodeEntry
          ACCESS  not-accessible
          STATUS  mandatory
          DESCRIPTION
                  “Individual resource group/node state descriptions”
          INDEX           { resGroupNodeGroupId, resGroupNodeId }
          ::= { resGroupNodeTable 1 }
--
  ResGroupNodeEntry::= SEQUENCE {
          resGroupNodeGroupIdINTEGER,
          resGroupNodeId  INTEGER,
          resGroupNodeStateINTEGER
  }

--
--B) The Resource Group Id
--
--
  resGroupNodeGroupId OBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The ID of the resource group”
          ::= { resGroupNodeEntry 1 }

--
--C) The Participating Node Id
--
--
  resGroupNodeId  OBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “Node ID of node located within resource group”
          ::= { resGroupNodeEntry 2 }
```

```
--
--D) The Resource Group Node State
--
--
  resGroupNodeStateOBJECT-TYPE
          SYNTAX  INTEGER {
                          online(2),
                          offline(4),
                          acquiring(16),
                          releasing(32),
                          error(64),
                          unknown(8)
                  }
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The State of the Resource Group”
          ::= { resGroupNodeEntry 3 }

  trapRGState       TRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ resGroupNodeGroupId, resGroupNodeId,
                           resGroupNodeState }
          DESCRIPTION
                  “Fires whenever a resource group changes
                  state on a particular node.”
          ::= 23

--
--
-- XIII. The clsmuxpd group
--
--A) clsmuxpdGets
--        Incremented on each get request.
--
  clsmuxpdGetsOBJECT-TYPE
          SYNTAX  Counter
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “Number of get requests received”
          ::= { clsmuxpd 1 }
--
--B) clsmuxpdGetNexts
--        Incremented on each get-next request.
--
  clsmuxpdGetNextsOBJECT-TYPE
          SYNTAX  Counter
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “Number of get-next requests received”
          ::= { clsmuxpd 2 }
--
--C) clsmuxpdSets
--        Incremented on each set request.
--
  clsmuxpdSetsOBJECT-TYPE
          SYNTAX  Counter
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “Number of set requests received”
          ::= { clsmuxpd 3 }
--
```

```
--D) clsmuxpdTraps
--        Incremented after a trap is generated.
--
  clsmuxpdTrapsOBJECT-TYPE
          SYNTAX  Counter
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “Number of traps sent”
          ::= { clsmuxpd 4 }
--
--E) clsmuxpdErrors
--        Incremented after an error occurs.
--
  clsmuxpdErrorsOBJECT-TYPE
          SYNTAX  Counter
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “Number of errors encountered”
          ::= { clsmuxpd 5 }
--
--F) clsmuxpdVersion
--        Version number of clsmuxpd program.
--
  clsmuxpdVersionOBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “Version of clsmuxpd program”
          ::= { clsmuxpd 6 }
--
--
--
-- XIV. The event group
--     This is a list of the last one thousand events called
--     by the Cluster Manager. This list is used for tracking
--     cluster event history.
--
--A) eventPtr
--        Points to the most recent event.
--
  eventPtrOBJECT-TYPE
          SYNTAX  Counter
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “Pointer to the most recent event”
          ::= { event 1 }
--
--B) The event table
--        This is a variable length table which is indexed by
--        a counter.  Useful for keeping history of events.
--
  eventTableOBJECT-TYPE
          SYNTAX  SEQUENCE OF EventType
          ACCESS  not-accessible
          STATUS  mandatory
          DESCRIPTION
                  “A series of cluster events”
          ::= { event 2 }
--
  eventTypeOBJECT-TYPE
          SYNTAX  EventType
```

```
          ACCESS  not-accessible
          STATUS  mandatory
          INDEX   { nodeId, eventCount }
          ::= { eventTable 1 }
--
  EventType::= SEQUENCE {
          eventId         INTEGER,
          eventNodeIdINTEGER,
          eventNetIdINTEGER,
          eventTimeINTEGER,
          eventCountCOUNTER,
          eventNodeNameDisplayString
  }
--
--C) eventId
--        This field is returned by the cluster manager.
--
  eventId OBJECT-TYPE
          SYNTAX  INTEGER { swapAdapter(0), swapAdapterComplete(1),
                   joinNetwork(2), failNetwork(3),
                   joinNetworkComplete(4), failNetworkComplete(5),
                   joinNode(6), failNode(7),
                   joinNodeComplete(8), failNodeComplete(9),
                   joinStandby(10), failStandby(11),
                   newPrimary(12),
                   clusterUnstable(13), clusterStable(14),
                   configStart(15), configComplete(16),
                   configTooLong(17), unstableTooLong(18),
                   eventError(19), dareTopology(20),
                   dareTopologyStart(21), dareTopologyComplete(22),
                   dareResource(23), dareResourceRelease(24),
                   dareResourceAcquire(25), dareResourceComplete(26),
                   resourceGroupChange(27) }
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The cluster event”
          ::= { eventType 1 }
--
--D) eventNodeId
--        This field is returned by the cluster manager.
--
  eventNodeIdOBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The ID of the Node on which the event occurs”
          ::= { eventType 2 }
--
--E) eventNetId
--        This field is returned by the cluster manager.
--
  eventNetIdOBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The ID of the Network on which the event occurs”
          ::= { eventType 3 }
--
--F) eventTime
--        This field is an integer string returned by the gettimeofday()
--               library call and is updated whenever an event is received.
```

```
--
  eventTimeOBJECT-TYPE
          SYNTAX  COUNTER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The time at which the event occurred”
          ::= { eventType 4 }
--
--G) eventCount
--        This field is incremented whenever an event is received.
--
  eventCountOBJECT-TYPE
          SYNTAX  COUNTER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “A count of the event used for indexing into the table”
          ::= { eventType 5 }
--
--H) eventNodeName
--        This field is returned by the cluster manager.
--
  eventNodeNameOBJECT-TYPE
          SYNTAX  DisplayString
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  “The name of the Node on which the event occurs”
          ::= { eventType 6 }

--
-- State Event traps
--
  trapSwapAdapterTRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ nodeName, clusterName, addrLabel, eventCount }
          DESCRIPTION
          “Specified node generated swap adapter event”
          ::= 64

  trapSwapAdapterCompleteTRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ nodeName, clusterName, netName, eventCount }
          DESCRIPTION
          “Specified node generated swap adapter complete event”
          ::= 65

  trapJoinNetworkTRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ nodeName, clusterName, netName, eventCount }
          DESCRIPTION
          “Specified node has joined the network”
          ::= 66

  trapFailNetworkTRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ nodeName, clusterName, netName, eventCount }
          DESCRIPTION
          “Specified node generated fail network event”
          ::= 67

  trapJoinNetworkCompleteTRAP-TYPE
          ENTERPRISErisc6000clsmuxpd
          VARIABLES{ nodeName, clusterName, netName, eventCount }
```

```
        DESCRIPTION
        “Specified node generated join network complete event”
        ::= 68

trapFailNetworkCompleteTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Specified node generated fail network complete event”
        ::= 69

trapJoinNodeTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Specified node generated join node event”
        ::= 70

trapFailNodeTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Specified node generated fail join node event”
        ::= 71

trapJoinNodeCompleteTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Specified node generated join node complete event”
        ::= 72

trapFailNodeCompleteTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName , eventCount}
        DESCRIPTION
        “Specified node generated fail node complete event”
        ::= 73

trapJoinStandbyTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Specified node generated join standby event”
        ::= 74

trapFailStandbyTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Specified node has failed standby adapter”
        ::= 75

trapEventNewPrimaryTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, clusterPrimaryNodeName, eventCount}
        DESCRIPTION
        “Specified node has become the new primary”
        ::= 76

trapClusterUnstableTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Specified node generated cluster unstable event”
```

```
        ::= 77

trapClusterStableTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Specified node generated cluster stable event”
        ::= 78

trapConfigStartTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Configuration procedure has started for specified node”
        ::= 79

trapConfigCompleteTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Configuration procedure has completed for specified node”
        ::= 80

trapClusterConfigTooLongTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Specified node has been in configuration too long”
        ::= 81

trapClusterUnstableTooLongTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Specified node has been unstable too long”
        ::= 82

trapEventErrorTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Specified node generated an event error”
        ::= 83

trapDareTopologyTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Dynamic reconfiguration event for topology has been issued”
        ::= 84

trapDareTopologyStartTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Dynamic reconfiguration event for topology has started”
        ::= 85

trapDareTopologyCompleteTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Dynamic reconfiguration event for topology has completed”
        ::= 86
```

```
trapDareResourceTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Dynamic reconfiguration event for resource has been issued”
        ::= 87

trapDareResourceReleaseTRAP-TYPE
        ENTERPRISErisc6000clsmuxpd
        VARIABLES{ nodeName, clusterName, netName, eventCount }
        DESCRIPTION
        “Dynamic reconfiguration event for resource has been released”
        ::= 88
```

HACMP for AIX Object Definitions mandatory

```
-- object definitions compiled from RISC6000CLSMUXPD-MIB { iso 3 6 1 4 1  }

ibm                 enterprises.2
ibmAgents           ibm.3
aix                 ibmAgents.1
aixRISC6000         aix.2
risc6000agents      aixRISC6000.1
risc6000clsmuxpd    risc6000agents.5
cluster             risc6000clsmuxpd.1
node                risc6000clsmuxpd.2
address             risc6000clsmuxpd.3
network             risc6000clsmuxpd.4
clstrmgr            risc6000clsmuxpd.5
cllockd             risc6000clsmuxpd.6
clinfo              risc6000clsmuxpd.7
application         risc6000clsmuxpd.8
clsmuxpd            risc6000clsmuxpd.9
event               risc6000clsmuxpd.10
resmanager          risc6000clsmuxpd.11

clusterId           cluster.1       INTEGER         read-only       mandatory
clusterName         cluster.2       DisplayString   read-only       mandatory
clusterConfiguration cluster.3      DisplayString   read-only       deprecated
clusterState        cluster.4       INTEGER         read-only       mandatory
clusterPrimary      cluster.5       INTEGER         read-only       mandatory
clusterLastChange   cluster.6       INTEGER         read-only       mandatory
clusterGmtOffset    cluster.7       INTEGER         read-only       mandatory
clusterSubState     cluster.8       INTEGER         read-only       mandatory
clusterNodeName     cluster.9       DisplayString   read-only       mandatory
clusterPrimaryNodeName cluster.10     DisplayString   read-only       mandatory
clusterNumNodes     cluster.11      INTEGER         read-only       mandatory
clusterNodeId       cluster.12      INTEGER         read-only       mandatory
nodeTable           node.1          Aggregate       not-accessible  mandatory
nodeEntry           nodeTable.1     Aggregate       not-accessible  mandatory
nodeId              nodeEntry.1     INTEGER         read-only       mandatory
nodeState           nodeEntry.2     INTEGER         read-only       mandatory
nodeNumIf           nodeEntry.3     INTEGER         read-only       mandatory
nodeName            nodeEntry.4     DisplayString   read-only       mandatory
addrTable           address.1       Aggregate       not-accessible  mandatory
addrEntry           addrTable.1     Aggregate       not-accessible  mandatory
addrNodeId          addrEntry.1     INTEGER         read-only       mandatory
addrAddress         addrEntry.2     IpAddress       read-only       mandatory
addrLabel           addrEntry.3     DisplayString   read-only       mandatory
addrRole            addrEntry.4     INTEGER         read-only       mandatory
addrNetId           addrEntry.5     INTEGER         read-only       mandatory
addrState           addrEntry.6     INTEGER         read-only       mandatory
addrActiveNode      addrEntry.7     INTEGER         read-only       mandatory
oldAddrActiveNode   addrEntry.8     INTEGER         not-accessible  mandatory
netTable            network.1       Aggregate       not-accessible  mandatory
```

```
netEntry              netTable.1       Aggregate       not-accessible  mandatory
netNodeId             netEntry.1       INTEGER         read-only       mandatory
netId                 netEntry.2       INTEGER         read-only       mandatory
netName               netEntry.3       DisplayString   read-only       mandatory
netAttribute          netEntry.4       INTEGER         read-only       mandatory
netState              netEntry.5       INTEGER         read-only       mandatory
clstrmgrTable         clstrmgr.1       Aggregate       not-accessible  mandatory
clstrmgrEntry         clstrmgrTable.1  Aggregate       not-accessible  mandatory
clstrmgrNodeId        clstrmgrEntry.1  INTEGER         read-only       mandatory
clstrmgrVersion       clstrmgrEntry.2  DisplayString   read-only       mandatory
clstrmgrStatus        clstrmgrEntry.3  INTEGER         read-write      mandatory
cllockdTable          cllockd.1        Aggregate       not-accessible  mandatory
cllockdEntry          cllockdTable.1   Aggregate       not-accessible  mandatory
cllockdNodeId         cllockdEntry.1   INTEGER         read-only       mandatory
cllockdVersion        cllockdEntry.2   DisplayString   read-only       mandatory
cllockdStatus         cllockdEntry.3   INTEGER         read-write      mandatory
clinfoTable           clinfo.1         Aggregate       not-accessible  mandatory
clinfoEntry           clinfoTable.1    Aggregate       not-accessible  mandatory
clinfoNodeId          clinfoEntry.1    INTEGER         read-only       mandatory
clinfoVersion         clinfoEntry.2    DisplayString   read-only       mandatory
clinfoStatus          clinfoEntry.3    INTEGER         read-write      mandatory
appTable              application.1    Aggregate       not-accessible  mandatory
appEntry              appTable.1       Aggregate       not-accessible  mandatory
appNodeId             appEntry.1       INTEGER         read-only       mandatory
appPid                appEntry.2       INTEGER         read-only       mandatory
appName               appEntry.3       DisplayString   read-only       mandatory
appVersion            appEntry.4       DisplayString   read-only       mandatory
appDescr              appEntry.5       DisplayString   read-only       mandatory
resGroupTable         resmanager.1     Aggregate       not-accessible  mandatory
resGroupEntry         resGroupTable.1  Aggregate       not-accessible  mandatory
resGroupId            resGroupEntry.1  INTEGER         read-only       mandatory
resGroupName          resGroupEntry.2  DisplayString   read-only       mandatory
resGroupPolicy        resGroupEntry.3  INTEGER         read-only       mandatory
resGroupUserPolicyName resGroupEntry.4  DisplayString   read-only       mandatory
resGroupNumResources  resGroupEntry.5  INTEGER         read-only       mandatory
resGroupNumNodes      resGroupEntry.6  INTEGER         read-only       mandatory
resTable              resmanager.2     Aggregate       not-accessible  mandatory
resEntry              resTable.1       Aggregate       not-accessible  mandatory
resourceGroupId       resEntry.1       INTEGER         read-only       mandatory
resourceId            resEntry.2       INTEGER         read-only       mandatory
resourceName          resEntry.3       DisplayString   read-only       mandatory
resourceType          resEntry.4       INTEGER         read-only       mandatory
resGroupNodeTable     resmanager.3     Aggregate       not-accessible  mandatory
resGroupNodeEntry     resGroupNodeTable.1 Aggregate       not-accessible  mandatory
resGroupNodeGroupId   resGroupNodeEntry.1 INTEGER         read-only       mandatory
resGroupNodeId        resGroupNodeEntry.2 INTEGER         read-only       mandatory
resGroupNodeState     resGroupNodeEntry.3 INTEGER         read-only       mandatory
clsmuxpdGets          clsmuxpd.1       Counter         read-only       mandatory
clsmuxpdGetNexts      clsmuxpd.2       Counter         read-only       mandatory
clsmuxpdSets          clsmuxpd.3       Counter         read-only       mandatory
clsmuxpdTraps         clsmuxpd.4       Counter         read-only       mandatory
clsmuxpdErrors        clsmuxpd.5       Counter         read-only       mandatory
clsmuxpdVersion       clsmuxpd.6       DisplayString   read-only       mandatory
eventPtr              event.1          Counter         read-only       mandatory
eventTable            event.2          Aggregate       not-accessible  mandatory
eventType             eventTable.1     Aggregate       not-accessible  mandatory
eventId               eventType.1      INTEGER         read-only       mandatory
eventNodeId           eventType.2      INTEGER         read-only       mandatory
eventNetId            eventType.3      INTEGER         read-only       mandatory
eventTime             eventType.4      COUNTER         read-only       mandatory
eventCount            eventType.5      COUNTER         read-only       mandatory
eventNodeName         eventType.6      DisplayString   read-only       mandatory
```

SNMP Operations

This section briefly describes the available AIX SNMP commands and functions you can use to query a MIB. See the AIX documentation for detailed information.

SNMP Commands

Several functions are available for AIX SNMP operations. Most are available as options to use with the **snmpinfo** command. This command requests or modifies values for one or more MIB variables for an SNMP agent. This command can take the following general formats:

```
snmpinfo -m get
snmpinfo -m next
snmpinfo -m set
snmpinfo -m dump
```

The following list describes the options.

get — Requests information about one or more MIB variables from an SNMP agent.

next — Requests information from an SNMP agent about the instances following the specified instances. This option makes it possible to obtain MIB values without knowledge of the instance qualifiers.

set — Modifies values for one or more MIB variables for an SNMP agent. Only a few MIB variables are designated read-write. The agent that manages the MIB database may take various actions as a result of any modification of MIB variables.

dump — Traverses the entire MIB tree of a given agent. If a group is passed in as the Variable parameter, the command traverses that specified path of the MIB tree.

The **snmpinfo** command also has a debug facility that dumps information for transmitted and received packets. See the AIX InfoExplorer facility for a detailed description of this command and its set of command line options.

The **snmpinfo** command can be mapped to the **snmp_get**, **snmp_next**, and **snmp_set** commands. The SNMP requests are issued to the specified SNMP agent by the **snmpinfo** command. If you use these mapped SNMP commands, use the syntax as described in the InfoExplorer facility.

These commands are useful in diagnosing problems with Clinfo clients, **clinfo**, or **clsmuxpd**. Problems with the MIB, though most likely caused by **clsmuxpd** operation or configuration problems, will affect Clinfo clients.

SNMP Traps

SNMP traps are sent by the HACMP SMUX daemon, **clsmuxpd**. Traps signal a change of state (cluster, cluster substate, address, network, node, newPrimary, adapterSwap, applications). All traps contain the name of the object whose state is transitioning, as well as the new state of the object. For applications, it also includes the nodeid of the node running the application and the PID if the application is active (or 0 if the application is down).

Following is a list of SNMP traps, with references to the location within the MIB definition (earlier in this appendix) where details are listed about each trap. The following SNMP traps can also be found in the source file **hacmp.my**.

SNMP Trap	For more information, see:
Trap 10 (TrapClusterState)	page A-3
Trap 11 (TrapClusterSubState)	page A-4
Trap 12 (TrapNodeState)	page A-6
Trap 13 (TrapNetworkState)	page A-8
Trap 14 (TrapAddressState)	page A-8
Trap 15 (TrapNewPrimary)	page A-4
Trap 16 (TrapAppState)	page A-14
Trap 17 (TrapAdapterSwap)	page A-8
Trap 19 (TrapAddressTakeover)	page A-8
Resource Group Traps	page A-15
State Event Traps	page A-22

Index

D

E

H

T

U

Vos remarques sur ce document / Technical publication remark form

Titre / Title : Bull HACMP 4.4 Programming Client Applications

Nº Reférence / Reference Nº : 86 A2 60KX 02

Daté / Dated : August 2000

ERREURS DETECTEES / ERRORS IN PUBLICATION

AMELIORATIONS SUGGEREES / SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Vos remarques et suggestions seront examinées attentivement.
Si vous désirez une réponse écrite, veuillez indiquer ci-après votre adresse postale complète.

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please furnish your complete mailing address below.

NOM / NAME : _____ Date : _____

SOCIETE / COMPANY : _____

ADRESSE / ADDRESS : _____

_____

Remettez cet imprimé à un responsable BULL ou envoyez-le directement à :

Please give this technical publication remark form to your BULL representative or mail to:

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Technical Publications Ordering Form

Bon de Commande de Documents Techniques

To order additional publications, please fill up a copy of this form and send it via mail to:
Pour commander des documents techniques, remplissez une copie de ce formulaire et envoyez-la à :

BULL CEDOC
ATTN / MME DUMOULIN
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Managers / Gestionnaires :
Mrs. / Mme : **C. DUMOULIN** +33 (0) 2 41 73 76 65
Mr. / M : **L. CHERUBIN** +33 (0) 2 41 73 63 96

FAX : +33 (0) 2 41 73 60 19
E–Mail / Courrier Electronique : srv.Cedoc@franp.bull.fr

Or visit our web site at: / Ou visitez notre site web à:
`http://www–frec.bull.com` (PUBLICATIONS, Technical Literature, Ordering Form)

CEDOC Reference # Nº Référence CEDOC	**Qty** Qté	**CEDOC Reference #** Nº Référence CEDOC	**Qty** Qté	**CEDOC Reference #** Nº Référence CEDOC	**Qty** Qté
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
[_ _] : **no revision number means latest revision** / pas de numéro de révision signifie révision la plus récente					

NOM / NAME : ______________________________ Date : __________

SOCIETE / COMPANY : ______________________________

ADRESSE / ADDRESS : ______________________________

____

PHONE / TELEPHONE : ____________________ FAX : ____________________

E–MAIL : ______________________________

For Bull Subsidiaries / Pour les Filiales Bull :
Identification: ______________________________

For Bull Affiliated Customers / Pour les Clients Affiliés Bull :
Customer Code / Code Client : ______________________________

For Bull Internal Customers / Pour les Clients Internes Bull :
Budgetary Section / Section Budgétaire : ______________________________

For Others / Pour les Autres :
Please ask your Bull representative. / Merci de demander à votre contact Bull.

PLACE BAR CODE IN LOWER LEFT CORNER

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
86 A2 60KX 02

Bull

Utiliser les marques de découpe pour obtenir les étiquettes.
Use the cut marks to get the labels.

AIX
HACMP 4.4
Programming
Client Applications

86 A2 60KX 02

AIX
HACMP 4.4
Programming
Client Applications

86 A2 60KX 02

AIX
HACMP 4.4
Programming
Client Applications

86 A2 60KX 02