ULTRIX

Guide to the Location Broker

Order Number: AA-PBKSA-TE
June 1990

Product Version: ULTRIX Version 4.0 or higher

This manual describes the administration of the Location Broker, a component of DECrpc. Location Broker software provides name service runtime support for distributed application programs that use remote procedure calls. These programs can include both user applications and system facilities.

digital equipment corporation
maynard, massachusetts

Contents

Glossary

Examples

Figures

Tables

About This Manual

This manual describes the administration of the Location Broker, a component of the Digital Remote Procedure Call (DECrpc) Version 1.0, which is based on and is compatible with Apollo's Network Computing System (NCS). Location Broker software provides name service runtime support for distributed application programs that use remote procedure calls. These programs can include both user applications and system facilities.

This manual is a new manual in the ULTRIX documentation set. The manual is based on the manual *Managing NCS Software* from the Apollo Systems Division of Hewlett Packard.

Audience

This manual is for system administrators who are setting up the Location Broker and who are administrating systems that are running distributed applications. It explains how to establish and maintain runtime support for distributed applications.

Organization

This manual contains three chapters, a glossary, and an index.

Chapter 1 describes the basic concepts of the Location Broker.

Chapter 2 describes `lb_admin`, the location broker administrative tool, and its use.

Chapter 3 describes how to set up and run the Location Broker daemons on homogeneous networks and networks that include Apollo systems. It also describes how to maintain the Location Brokers.

If you want to set up the Location Broker immediately, you can follow the procedures in Chapter 3 without reading any of the other chapters.

Related Documentation

For more information on topics related to DECrpc, see the following documents:

DECrpc Programming Guide

This book is a reference manual for programmers developing distributed applications. It provides programming information and examples of programs using remote procedure calls, including calls to the Location Broker.

ULTRIX Reference Pages

The *ULTRIX Reference Pages* describe the commands and special files referred to in this manual, and the library routines described in the *DECrpc Programming Guide.*

Conventions

The following conventions are used in this guide:

`special`	In text, each mention of a specific command, option, partition, pathname, directory, or file is presented in this type.
`command(x)`	In text, cross-references to command documentation include the section number in the reference manual where the command is documented. For example: See the `cat`(1) command. This indicates that you can find the material on the `cat` command in Section 1 of the *ULTRIX Reference Pages*.
variable	In syntax descriptions, this type indicates terms that are variable.
`literal`	In syntax descriptions, this type indicates terms that are constant and must be typed just as they are presented.
[ ]	In syntax descriptions, brackets indicate terms that are optional.
. . .	In syntax descriptions, a horizontal ellipsis indicates that the preceding item can be repeated one or more times.
UPPERCASE	The ULTRIX operating system differentiates between lowercase and uppercase characters. Enter uppercase characters only where specifically indicated by an example or a syntax line.
`example`	In examples, computer output text is printed in this type.
`example`	In examples, user input is printed in this bold type.
new term	In text, new terms are introduced in this bold type.
`$`	This is the default user prompt in multiuser mode.
`#`	This is the default superuser prompt.
. . .	A vertical ellipsis indicates that a portion of an example that would normally be present is not shown.

Location Broker Concepts 1

The Location Broker is a component of DECrpc, which is based on and is compatible with the Network Computing System (NCS), a set of tools for heterogeneous distributed computing. This chapter introduces the Location Broker and describes the role that it plays in distributed applications.

The Location Broker is a name service that provides clients with information about the locations of objects and interfaces. An object is an entity accessed by well-defined operations. A file, a directory, a database, a serial line, a printer, and a processor can all be objects. Each interface is a set of operations that can be applied to any of those objects. Every object has a type. For example, you can classify printer queues as objects of the type **printqueue**, accessed through a **printqueue_ops** interface that includes operations to add, delete, and list jobs in the queues.

DECrpc identifies every object, type, and interface by a Universal Unique Identifier (UUID). A UUID is defined as a 16-byte quantity identifying the host on which the UUID is created and the time at which it is created. Six bytes identify the time, two are reserved, and eight identify the host.

The `uuid_gen`(1ncs) utility provided in DECrpc generates UUIDs as text strings or as data structures defined in C syntax. The string representation used by the NIDL Compiler, a tool for developing distributed applications, and by DECrpc utilities consists of 28 hexadecimal characters arranged as in this example:

```
3a2f883c4000.0d.00.00.fb.40.00.00.00
```

Servers register with the Location Broker their socket addresses and the objects and interfaces to which they provide access. Clients issue requests to the Location Broker for the locations of objects and interfaces they wish to access. The broker then returns database entries that match an object, type, interface, or some combination of these, as specified in the request.

The Location Broker also implements the RPC message-forwarding mechanism. If a client sends a request for an interface to the Location Broker forwarding port on a host, the broker automatically forwards the request to the appropriate server on the host.

1.1 Location Broker Software

The Location Broker consists of the following interrelated components:

- The Local Location Broker
- The Global Location Broker
- The Location Broker Client Agent

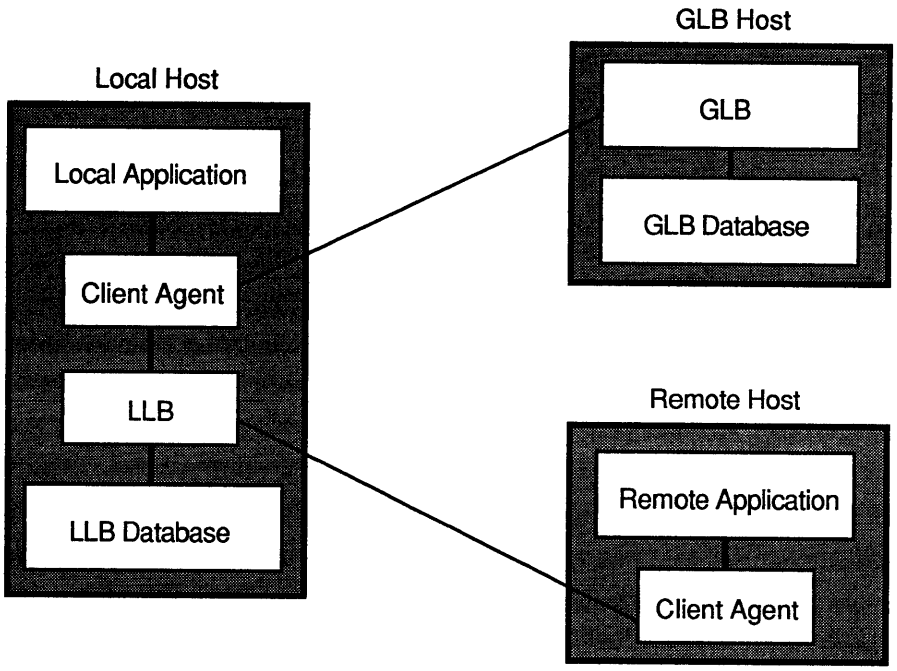
The **Local Location Broker (LLB)** is an RPC server that maintains a database of information about objects and interfaces located on the local host. The LLB runs as the daemon `llbd`. The LLB provides access to its database for application

programs and also provides the Location Broker forwarding service. An LLB must run on any host that runs RPC servers.

The **Global Location Broker (GLB)** is an RPC server that maintains information about objects and interfaces throughout the network or internet. The GLB runs as the daemon `nrglbd`.

The **Location Broker Client Agent** is a set of library routines that application programs call to access LLB and GLB databases. Any client that uses Location Broker routines is actually making calls to the Client Agent. The Client Agent interacts with the LLBs and the GLB to provide access to their databases. Figure 1-1 shows the relationships among application programs, the Location Broker components, and the Location Broker databases.

Figure 1-1: Location Broker Software



1.1.1 Location Broker Data

Each entry in a Location Broker database contains information about an object, an interface, and the location of a server that exports the interface to the object. Table 1-1 lists the fields in a database entry. Use the `lb_admin` command, described in Chapter 2, to view database entries.

Table 1-1: Location Broker Database Entry

Field	Description
Object UUID	The unique identifier of the object
Type UUID	The unique identifier that specifies the type of the object
Interface UUID	The unique identifier of the interface to the object
Flag	A flag that indicates whether the object is global (and therefore should be registered in the GLB database) or local (and therefore should be registered in the LLB database)
	All global registrations are also registered with the LLB on the host where the server registers.
Annotation	Sixty-four characters of user-defined information
Socket address length	The length of the socket address field
Socket address	The location of the server that exports the interface to the object

Because each database entry contains one object UUID, one interface UUID, one type UUID, and one socket address, a Location Broker database must have an entry for each possible combination of object, interface, type, and socket address.

Thus, the database must have 10 entries for a server that:

- Listens on two sockets, `socket_a` and `socket_b`
- Exports `interface_1` for `object_x`, `object_y`, and `object_z`
- Exports `interface_2` for `object_p` and `object_q`
- Has only one type for `object_p` and `object_q`

You can look up Location Broker information by using any combination of the object UUID, type UUID, and interface UUID as keys. You can also request the information from the GLB database or from a particular LLB database. Therefore, you can obtain information about all objects of a specific type, all hosts with a specific interface to an object, or even all objects and interfaces at a specific host. For example, you could find the addresses of all remotely available array processors by looking up all entries with the **arrayproc** type.

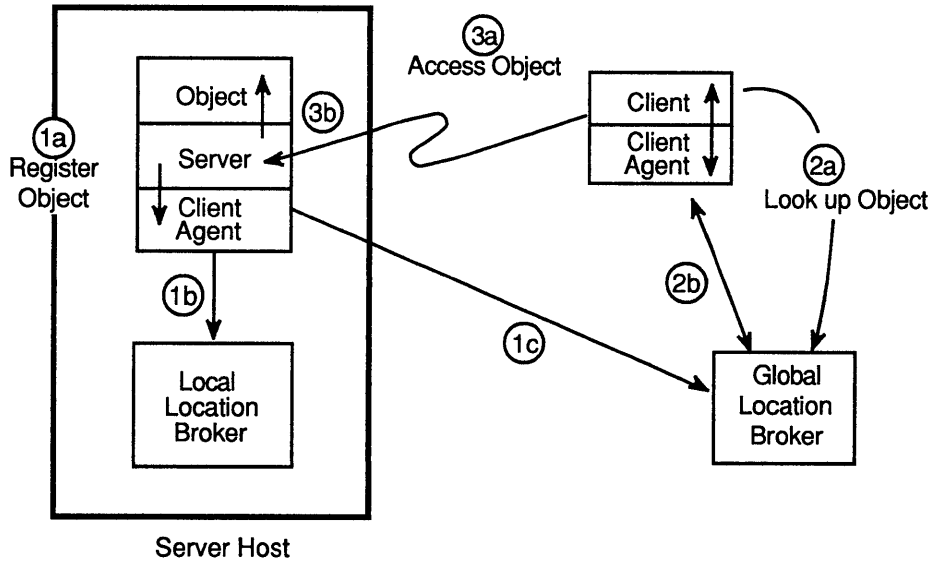
1.1.2 Location Broker Registrations and Lookups

The Location Broker Client Agent is a set of library routines that applications use to access and modify the LLB and GLB databases. When a program issues any Location Broker call, the call actually goes to the local host Client Agent. The Client Agent then does the work to add, delete, or look up information in the appropriate Location Broker database.

Figure 1-2 illustrates a typical case in which a client requires a particular interface to a particular object but does not know the location of a server exporting the interface to the object. In this figure, an RPC server registers itself with the Location Broker

by calling the Client Agent in its host (1a). The Client Agent registers the server with the LLB at the server host (1b) and with the GLB (1c). To locate the server, the client issues a Location Broker lookup call (2a). The Client Agent on the client host sends the lookup request to the GLB, which returns it through the Client Agent to the client (2b). The client can then use RPC calls to communicate directly with the located server (3a, 3b).

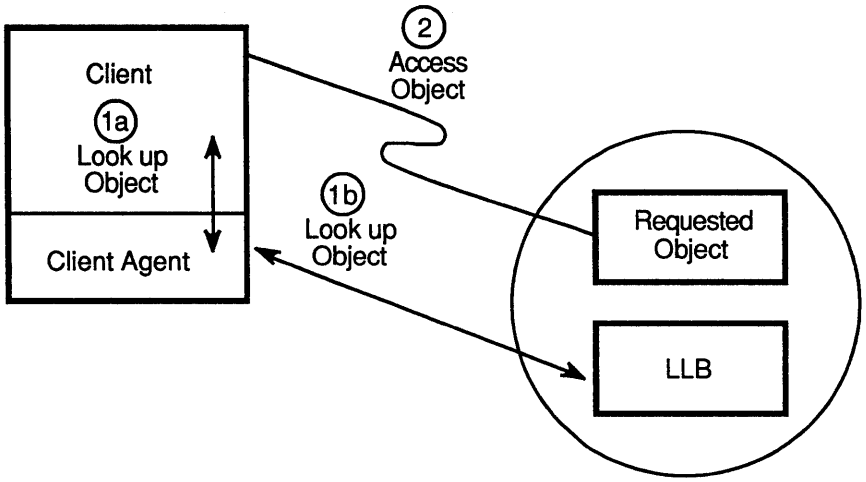
Figure 1-2: Client Agents and Location Brokers



ZK-0086U-R

A client might know the host where the object is located without knowing the port number used by the server. The client can specify this information in its lookup call. In this case, the Client Agent directly interrogates the LLB of the remote host, as illustrated in Figure 1-3.

Figure 1-3: Client Agent Doing a Lookup at a Known Host



1.2 The Local Location Broker

The LLB runs as the `llbd` daemon. It has two major functions:

- It maintains a database of the objects and interfaces that are exported by servers running on the host.
- It acts as a forwarding agent for requests.

Although it is recommended that you run an `llbd` daemon on every host, the daemon is required only on hosts that run RPC servers.

1.2.1 The Local Database

The LLB database provides location information about interfaces on the local host. This information is used by both local and remote applications. To look up information in an LLB database, an application queries the LLB through a client agent. (See Figure 1-1.)

1.2.2 The LLB Forwarding Agent

The forwarding facility of the LLB eliminates the need for a client to know the specific port that a server uses and thereby helps to conserve well-known ports. The forwarding agent listens on one well-known port for each address family. It forwards any messages that it receives to the local server that exports the requested object.

Forwarding is particularly useful when the requestor of a service already knows the host where the server is running. Such a server can use a dynamically assigned opaque port and register only with the LLB at its local host, not with the GLB. To access the server, a client needs only to specify the object, the interface, and the host, but not a specific port.

1.3 The Global Location Broker

The GLB manages information about the objects and interfaces that are available to users on the network. An RPC server registers itself with the Location Broker by calling the Client Agent on its host. The Client Agent adds the registration information to the LLB database at the server host and also sends the information to the GLB.

When a client requires the services of a particular interface, it issues a Location Broker lookup call to the Client Agent on its host. The Client Agent sends the lookup request to the GLB. The GLB extracts the required information from its database and returns it, through the Client Agent, to the Client. The client can then use RPC system calls to communicate directly with the located server.

When the GLB receives calls, either from a server or from `lb_admin`, to register or unregister an object, the GLB daemon updates the GLB database.

lb_admin: The Location Broker Administrative Tool 2

The Location Broker administrative tool, `lb_admin`, allows you to inspect or modify the contents of a Location Broker database. Use `lb_admin` to look up information, add new entries, and delete existing entries in any LLB or GLB database.

The `lb_admin` tool is useful both for inspecting the contents of the Location Broker databases and for correcting database errors. For example, if a server starts while the `nrglbd` is not running, you can manually enter the information for the server in the GLB database. Similarly, if a server terminates abnormally without unregistering itself, you can use `lb_admin` to manually remove the server entry from the GLB database, the LLB database, or both.

2.1 Starting lb_admin

To start `lb_admin` on an ULTRIX system, enter this command:

```
$ /etc/ncs/lb_admin
lb_admin:
```

Table 2-1 lists the options to the `lb_admin` command.

Table 2-1: lb_admin Options

Option	Function
–nq	Do not query for verification of wildcard expansions in `unregister` operations. This option has no effect on the `clean` command. However, if you use wildcards for object and type, and specify interface and IP address, `lb_admin` deletes all interfaces of the same type as that of the interface specified.
–version	Display the version of the Network Computing Kernel (NCK) that this `lb_admin` belongs to, but do not start `lb_admin`. (NCK is part of the Network Computing System.)

At the `lb_admin:` prompt, enter any of the `lb_admin` commands, described in Section 2.2, or on the `lb_admin`(1ncs) reference page.

2.2 The lb_admin Command-Line Interface

Table 2-2 describes the commands that you can enter at the `lb_admin` prompt. Table 2-3, lb_admin Fields and Arguments, contains descriptions of the arguments and values for the commands. Section 2.4 contains examples of the commands.

Table 2-2: lb_admin Commands

Command	Syntax and Function
`add`	`a[dd]` *object type interface location annotation* [*flag*] Synonym for `register`.
`clean`	`c[lean]` Find and delete obsolete entries in the current database The `lb_admin` utility tries to contact each registered server. If the server responds, `lb_admin` leaves its registration entry intact. If the server does not respond, `lb_admin` tries to look up its registration in the LLB database at the host where the server is located, tells you the result of this lookup, and asks whether you want to delete the entry. The `lb_admin` utility queries even if you used the `nq` option. If a server responds but its UUIDs do not match the entry in the database, `lb_admin` tells you this result and asks whether you want to delete the entry. Section 2.4 includes an example of deleting an interface from the database.
`delete`	`d[elete]` *object type interface location* Synonym for `unregister`.
`help`	`h[elp]` [*command*] or `?` [*command*] Without a command name, lists the `lb_admin` commands. With a command name, displays a description of that command.
`lookup`	`l[ookup]` *object type interface* Look up all entries with matching *object*, *type*, and *interface* fields. You can use asterisks as wildcards in any of the arguments. If you enter all arguments as wildcards or enter no arguments, `lookup` displays the entire database.
`quit`	`q[uit]` Exit `lb_admin` and return to the shell.

Table 2-2: (continued)

Command	Syntax and Function
`register` `add`	`r[egister]` *object type interface location annotation* [*flag*] `a[dd]` *object type interface location annotation* [*flag*] Register the specified entry. You must supply the *object*, *type*, *interface*, *location*, and *annotation* fields. Wildcards are allowed in the *object*, *type*, and *interface* fields. You can enter a string of up to 64 characters to annotate the entry. Double quotation marks delimit a string that contains a space, or a null annotation. Use a backslash to escape double quotation marks in the annotation string. Values for `flag` are `local` or `global`. A `local` entry indicates the entry should be marked for local registration only, the default. A `global` entry indicates the entry should be marked for registration in both the LLB and GLB databases. The field is stored with the entry and does not affect where the entry is registered. The `register` command registers an entry only in the specified database. If you set the broker to the LLB and then register an entry with the `global` flag, the entry is registered only in the LLB database, and you must use a separate command to register it in the GLB database.
`set_broker`	`s[et_broker]` [*broker_switch*] *location* Set the host for the current LLB or GLB. If you specify `global` as the `broker_switch`, `set_broker` sets the current GLB. Otherwise, it sets the current LLB. Issue `use_broker`, not this command, to determine whether subsequent operations access the LLB or the GLB.
`set_timeout`	`set_t[imeout]` [*short* / *long*] Set the timeout period used by `lb_admin` for all of its operations. With an argument of short or long, `set_timeout` sets the timeout accordingly. With no argument, it displays the current timeout value.

Table 2-2: (continued)

Command	Syntax and Function
`unregister` `delete`	`u[nregister]` *object type interface location* `d[elete]` *object type interface location* Unregister all entries that match the specified arguments. An asterisk in the *object*, *type*, or *interface* field matches any other entry with an asterisk in that field. Unless you suppress queries by specifying the `nq` option to `lb_admin`, `unregister` asks you whether to delete each matching entry. Respond y[es] to delete all entries. Respond n[o] to leave the entry in the database. Respond g[o] to delete all remaining database entries that match, with no query. Respond q[uit] to terminate the unregister operation, without deleting any more entries. If you use wildcards, `lb_admin` prompts you to verify that you wish to delete each entry. With the `nq` option, if you use wildcards for object and type, and specify interface and IP address, `lb_admin` deletes all interfaces of the specified type.
`use_broker`	`us[e_broker]` *broker_switch* Specify whether to look up, register, or unregister entries in the current LLB database or in the current GLB database. Use `set_broker` to set the particular LLB or GLB to be accessed.

2.3 lb_admin Fields and Arguments

Table 2-3 lists the values that you specify to `lb_admin` as arguments in the command interface.

Table 2-3: lb_admin Fields and Arguments

Field	Argument
object type interface	Object, type, and interface UUIDs have the following format (in which n is a hexadecimal digit): `nnnnnnnnnnnn.nn.nn.nn.nn.nn.nn.nn` An asterisk is the wildcard for this argument.

Table 2-3: (continued)

Field	Argument
location	A socket address specifier consisting of an address family, a host name, and a port. The syntax is `family:name[port]`; for instance, `ip:cactus[104]`. DECrpc supports only the Internet address family, indicated as **ip**. The `name` argument is the host name, in a format suitable for the address family. The default is the local host. The `port` argument is an integer port number. The enclosing brackets are required if you are specifying the port field. If omitted, the default is `socket_$unspec_port`.
annotation	A string of up to 64 characters annotating the entry. In the command interface, you must use double quotation marks around any string that includes spaces, and you can embed a quotation mark in the string with a preceding backslash.
flag	In the command interface, specify as an argument either `local` or `global`. This flag marks a database entry to indicate whether that entry should be registered globally. However, it does not affect how the entry is registered when you use the `register` command. If omitted, the default is `local`.
broker_switch	In the command interface, specify `local` or `global`. This argument of a `set_broker` or `use_broker` command determines whether the LLB or the GLB is being specified. If omitted, the default is `local`.

2.4 Examples of lb_admin Commands

The examples in this section illustrate the `lb_admin` commands.

Although the `register` command example shows the command split across two lines for printing purposes, you must enter the entire command on a single line.

Example 2-1 shows the display from the `lb_admin help` command. Typing a question mark (?) at the `lb_admin` prompt produces the same display.

Example 2-1: The lb_admin Help Command

```
lb_admin: help
Known commands are:
        a[dd]           c[lean]         d[elete]
        r[egister]      u[nregister]    s[et_broker]
        set_[timeout]   us[e_broker]    l[ookup]
        e[xit]          q[uit]          h[elp]
        ?
```

Example 2-2 requests the use of the current GLB with the `use_broker` command and then displays the entire database with the `lookup` command with no arguments.

Example 2-2: Displaying the Contents of a Database

```
lb_admin: use_broker global
lb_admin: lookup
   Data from GLB replica: ip:cactus
------------
      object = *
        type = *
   interface = 43fd154a1696.02.82.b4.05.a0.00.00.00
"DCC on desert" @ ip:desert[1481] global
"DCC on cactus" @ ip:cactus[2455] global
"DCC on ramada" @ ip:ramada[3168] global
------------
      object = *
        type = *
   interface = 4460c9baef60.02.82.b4.05.a0.00.00.00
"dcc_server on cactus" @ ip:cactus[2936] global
------------
```

Examples 2-3 and 2-4 show `lb_admin register` commands that register an interface, `testregister`, in the local database. Both examples use the `lookup` command to display the database with the added interface. Example 2-3 shows the use of a wildcard for the interface and Example 2-4 uses a UUID for the interface.

Typically registration occurs from within a program. During test or debug situations, however, it may be convenient to manually register an interface without having to write code to do so.

Example 2-3: Registering an Interface (Wildcard for Interface)

```
lb_admin: register * * * ip:cactus testregister local
lb_admin: lookup
------------
      object = 333b91c50000.0d.00.00.87.84.00.00.00
        type = 333b91de0000.0d.00.00.87.84.00.00.00
   interface = 333b2e690000.0d.00.00.87.84.00.00.00
"non-replicated GLB" @ ip:cactus[1045]
------------
      object = *
        type = *
   interface = 4460c9baef60.02.82.b4.05.a0.00.00.00
"dcc_server on cactus" @ ip:cactus[3197] global
------------
      object = *
        type = *
   interface = *
"testregister" @ ip:cactus[0]
------------
```

Note

Example 2-4 shows the command line continued on a second line because of the short line length of the manual. You would, however, type the command on a single line.

Example 2-4: Registering an Interface (UUID for Interface)

```
lb_admin: register * * 4279729d556c.02.82.b4.05.a0.00.00.00
          ip:cactus testuuid local
lb_admin: lookup
------------
      object = *
        type = *
   interface = 4279729d556c.02.82.b4.05.a0.00.00.00
"testuuid" @ ip:cactus[0]
------------
lb_admin:
```

Example 2-5 shows `lb_admin` commands that list the current database, delete a specific interface, and then list the database again.

Example 2-5: Deleting (Unregistering) an Interface

```
lb_admin: lookup
------------
      object = 333b91c50000.0d.00.00.87.84.00.00.00
        type = 333b91de0000.0d.00.00.87.84.00.00.00
   interface = 333b2e690000.0d.00.00.87.84.00.00.00
"non-replicated GLB" @ ip:cactus[1045]
------------
      object = *
        type = *
   interface = 4460c9baef60.02.82.b4.05.a0.00.00.00
"dcc_server on cactus" @ ip:cactus[3197] global
------------
      object = *
        type = *
   interface = *
"testregister" @ ip:cactus[0]
------------
lb_admin: delete * * *  ip:cactus [1]
------------
      object = *
        type = *
   interface = *
"testregister" @ ip:cactus[0]
delete ? yes
lb_admin: lookup
------------
      object = 333b91c50000.0d.00.00.87.84.00.00.00
        type = 333b91de0000.0d.00.00.87.84.00.00.00
   interface = 333b2e690000.0d.00.00.87.84.00.00.00
"non-replicated GLB" @ ip:cactus[1045]
------------
      object = *
        type = *
   interface = 4460c9baef60.02.82.b4.05.a0.00.00.00
"dcc_server on cactus" @ ip:cactus[3197] global
------------
```

[1] A wildcard in a command field matches a corresponding interface field that also has an asterisk entry. This command, therefore, only deletes the interface for `testregister`, which is defined by an asterisk wildcard in each of its three fields.

Example 2-6 illustrates the `lb_admin clean` command that cleans the database of interfaces that do not respond.

Example 2-6: Deleting (Cleaning) Unresponding Interfaces

```
lb_admin: clean
working . . .
------------
      object = *
        type = *
   interface = 4279729d556c.02.82.b4.05.a0.00.00.00
"testregister" @ ip:cactus[0]
        Server not responding, but registered
        in remote llbd database.
        [using short timeouts] Delete? y
1 entries deleted of 1 entries processed
```

Administering the Location Brokers 3

This chapter provides guidelines for administering the Local and Global Location Broker in homogeneous networks and in networks that include Apollo systems.

3.1 Guidelines for Configuring the Location Brokers

The following sections provide guidelines for configuring the Local and Global Location Broker daemons.

3.1.1 Configuring the Local Location Broker Daemon

The following guidelines will help you to configure the Local Location Broker (LLB) daemon:

- Run the LLB daemon, `llbd`, on any host where a Global Location Broker (GLB) or other RPC-based server runs.
- Start the `llbd` from a system start-up script as described in Section 3.2.

Note

If your system contains more than one broadcast interface, DECrpc software uses only the broadcast interface that is associated with the `/bin/hostname` value.

3.1.2 Configuring the Global Location Broker Daemon

The following guidelines will help you to configure the Global Location Broker daemon.

Run the GLB daemon, `nrglbd`, on a single host in any network where RPC-based servers run.

Note

If the host running the `nrglbd` is taken off-line, any system that requires global location data will be unable to run distributed applications.

Section 3.3 describes procedures for restarting and maintaining the GLB and the GLB databases.

An `nrglbd` communicates with clients by means of Internet Protocols (IP). In an internet, one `nrglbd` can run on each network, but each `nrglbd` services only the hosts in its own network.

3.1.3 Configuring a Global Broker in Networks with Apollo Systems

On any network that contains Apollo nodes, it is preferable to run the replicatable GLB daemon, `glbd`, on an Apollo system, rather than `nrglbd` on a Digital system. The following guidelines will help you to configure the replicating GLB daemon on an Apollo system:

- If an Apollo system in the network is running a replicatable Global Location Broker daemon (`glbd`), use `glbd` on the Apollo system as the GLB daemon for the network. A replicated database provides increased performance and reliability.

  Refer to *Managing NCS Software* supplied with the Apollo system for procedures for administering the `glbd`.

- Because the `nrglbd` and `glbd` daemons do not interoperate, run only one or the other in a network.

- If an Apollo node is acting as a gateway between two networks, a `glbd` running on that node can serve clients on both networks. For increased performance and reliability, replicate the GLB on one or more other Apollo systems on either network.

- A `glbd` can communicate with another `glbd` only by means of Apollo Domain network communications protocols (DDS). Therefore, in an internet containing several GLB replicas, any gateways used by the GLB must support DDS.

3.2 Procedures for Starting Location Broker Daemons

Start the Location Brokers in the following order:

1. The LLB daemon, `llbd`
2. The GLB daemon, `nrglbd`
3. Any other RPC-based servers

The following sections describe the procedures for starting up the Local and Global Location Broker processes.

3.2.1 Starting the Local Broker Daemon

The `llbd` daemon is typically started at boot time in a system start-up script. It should run in the background, independently of login activity, for as long as the system is up.

Any host that runs an DECrpc server must also run an `llbd` daemon.

Example 3-1 shows the line in the ULTRIX start-up script, `/etc/rc.local`, that starts the LLB daemon.

Example 3-1: Starting the LLB Daemon at System Startup

```
/etc/ncs/llbd& echo -n ' llbd' > /dev/console
```

Example 3-2 shows the command for starting the LLB daemon manually on a running ULTRIX system.

Example 3-2: Starting the LLB Daemon Manually on a Running System

```
# /etc/ncs/llbd&
```

3.2.2 Starting the Global Location Broker Daemon

The GLB daemon is typically started at boot time in a system start-up script. It has no options and takes no arguments. It should run in the background independently of login activity for as long as the system is up.

Example 3-3 shows the line in the ULTRIX start-up script, `/etc/rc.local`, that starts up a GLBD daemon.

Example 3-3: Starting the GLB Daemon at System Startup

```
/etc/ncs/nrglbd&  echo -n ' nrglbd' > /dev/console
```

Example 3-4 shows the command for starting the GLB daemon manually on a running system.

Example 3-4: Starting the GLB Daemon Manually on a Running System

```
$ /etc/ncs/nrglbd&
```

3.3 Maintaining the Location Brokers

When an `llbd` daemon is started on a system running the ULTRIX operating system, it looks for the file, `/tmp/llbdbase.dat`. If it is found, the `llbd` automatically registers all the servers listed in the file.

When an `nrglbd` daemon is started on a system running the ULTRIX operating system, it looks for the file, `/etc/ncs/glbdbase.dat`. If it is found, the `nrglbd` automatically globally registers all the servers listed in the file.

The following sections provide information on changing the host running the GLB daemon and maintaining the location broker databases in case of system crash or downtime. The purpose of most of the procedures in the following sections is to provide up-to-date copies of the database files in case a host crashes or is taken off line.

3.3.1 Changing the GLB Host

To change the host running the GLB daemon, you need to perform these steps:

1. Copy the GLB database file, `/etc/ncs/glbdbase.dat` on an ULTRIX system, from the current GLB host to the new host.
2. Stop the running GLB daemon on the old host.
3. Start `nrglbd` on the new host, either manually or from the script, as described in Section 3.2.2
4. Shutdown the old GLB host.

Starting `nrglbd` on the new machine before shutting down the old GLB machine allows application servers that may be running on the old GLB host to unregister from the `nrglbd` database file on the new GLB host.

If this is not possible (due to a system crash or some other reason), run the `lb_admin clean` command on the new `nrglbd` host while the old `nrglbd` host is off line. The `lb_admin clean` command causes `lb_admin` to try to contact all registered servers. If a server is not responsive, `lb_admin` gives you the option of letting `lb_admin` send an unregistration request to the new `nrglbd` host.

3.3.2 Recovering from a Crash of the GLB Host

If the Global Location Broker host crashes and you restart `nrglbd` on the same system, all things should run as before as long as servers on other systems have not changed state. If, however, an application tries to contact a server that is no longer running or which has different port numbers, the application will fail. The application also will not see any new server registrations.

If a copy of `/etc/ncs/glbdbase.dat` is unavailable, you must create an up-to-date version of the file before restarting `glbd`. To do this, use `lb_admin` to query the `llbd` for registration data on every system running an RPC server and use `lb_admin` to register all servers with the GLB on the new host. Then, restart `nrglbd` as described in Section 3.2.

To make sure you have an up-to-date copy of `/etc/ncs/glbdbase.dat`, you could run a `cron`(8) task that copies the database file from the current `nrglbd` host to any other hosts that may want to run the `nrglbd` in the future. Because registrations and unregistrations of global data are normally done infrequently, this should ensure availability of a current copy.

To automatically start up `nrglbd` at a new host in case of a system crash, you could run a task that periodically makes a global lookup request. On failure, the task could fork off a child process to start a new `nrglbd` at another host on the local network.

3.3.3 Restarting a System Running Only the LLB

If all DECrpc servers are started from a file such as `/etc/rc.local`, and `/tmp` is cleared on system startup, then no problems should be seen when you restart a system running only the `llbd`. When the `llbd` restarts, it will rebuild a new `/tmp/llbdbase.dat` file.

However, if some DECrpc server is initiated by a user, and `/tmp` is not totally cleared on restart, or the `llbd` process itself is terminated (with `kill -9`), you may see problems when `llbd` initializes from the old `/tmp` files.

Old locally registered server information may exist that is no longer valid. In this case, however, only applications written to perform local lookups will be affected.

Any user may correct local `llbd` registrations with either the `lb_admin delete` or `clean` commands. Once the old registrations are removed, you can manually enter new entries or bring up RPC-based servers, or both.

The `nrglbd` is included in the context of server. When an `nrglbd` is brought up, it registers its interfaces with the `llbd`. If you change the running `nrglbd` from one host to another (by stopping the `nrglbd` daemon on one host and restarting it

on another), the host that was originally running the `nrglbd` retains its registration of the `nrglbd` interface. When a new RPC-based server registers on the host no longer running the `nrglbd`, The registration is forwarded to the new host running the `nrglbd`.

3.3.4 Terminating RPC Servers

Do not terminate registered RPC-based application servers (with `kill -9`), unless it is absolutely essential, because the servers' registration information will not be removed from the local or global database files. As a result, application client processes may try to bind to servers that are no longer available.

Instead, all DECrpc servers should be written to set a signal handler for process termination and initiate logic to unregister themselves from the LB or GLB brokers.

When a system is shutdown, the normal process of shutting down with the `shutdown`(8) command results in a process termination signal being sent to the running systems tasks. Each distributed application server should be prepared to process this signal and take appropriate unregistration steps.

Glossary

address family
: A set of communications protocols that use a common addressing mechanism to identify endpoints. The terms *address family* and *protocol family* are used synonymously in this manual.

Berkeley UNIX socket abstraction
: A network programming abstraction, developed by the University of California at Berkeley, that is independent of communications protocols and is based on the concept of sending and receiving datagrams.

broadcast
: To send a remote procedure call request to all hosts in a network.

broker
: A server that manages information resources. A Location Broker is a broker.

client
: A process that uses resources. In the context of this manual, a client is a program that makes remote procedure calls. A client imports one or more interfaces. *See also* **server.**

export an interface
: To provide the operations defined by an interface. A server exports an interface to a client. *See also* **import an interface.**

forward
: To dispatch a remote procedure call request to a server that exports the requested interface for the requested object. The Local Location Broker (LLB) forwards remote procedure calls that are sent to the LLB forwarding port on a server host.

Global Location Broker (GLB)
: A server that maintains global information about objects on a network or an internet. Part of the DECrpc Location Broker. The server runs as the **nrglbd** daemon.

host
: A computer that is attached to a network.

implement an interface
: Provide access to one or more objects. A server, a program that provides such access, accepts requests for operations in any of its interfaces. When it receives a request from a client, it executes the procedures that perform the operation and it sends a response to the client.

import an interface
: To request the operations defined by an interface. A client imports an interface from a server. *See also* **export.**

interface
: A set of operations. The Network Interface Definition Language defines interfaces.

interface UUID
: A Universal Unique Identifier (UUID) that permanently identifies a particular interface. Both the Remote Procedure Call (RPC) runtime library and the Location Broker use interface UUIDs to specify interfaces.

internet
: A set of two or more connected networks. The networks in an internet do not necessarily use the same communications protocol.

Local Location Broker (LLB)
: A server that maintains information about objects on the local host. The LLB also provides the Location Broker forwarding facility.

Location Broker
: A set of software including the Local Location Broker, the Global Location Broker, and the Location Broker Client Agent. The Location Broker maintains information about the locations of objects and interfaces.

Location Broker Client Agent
: Part of the Location Broker. Programs communicate with Global Location Brokers and Local Location Brokers by means of the Location Broker Client Agent.

manager
: A set of procedures that implement the operations in one interface for objects of one type. It is possible for a server to export several interfaces or to export an interface for several types of objects; each combination of interface and type has its own manager.

network
: Data transmission media through which computers can be connected.

Network Computing System (NCS)
: A set of software components that conform to the Network Computing Architecture. These components include the Remote Procedure Call runtime library, the Location Broker, and the Network Interface Definition Language (NIDL) Compiler. The DECrpc is based on NCS.

network ID
: Part of the network address that uniquely identifies a network. In the Internet, the number of bits that indicate the network ID are determined by the address class. In class A addresses, 7 bits indicate the network ID; in class B addresses, 14 bits indicate the network ID; in class C addresses, 22 bits indicate the network ID.

NIDL Compiler
: A tool that converts an interface definition written in Network Interface Definition Language (NIDL) into several program modules, including source code for client and server stubs. The NIDL Compiler accepts interface definitions written in the C syntax of NIDL; it generates C source code and header files.

Network Interface Definition Language(NIDL)
: A declarative language for the definition of interfaces. The NIDL syntax resembles the syntax of the C programming language.

object
: An entity that is manipulated by well-defined operations. Disk files, printers, and array processors are examples of objects. Objects are accessed though interfaces. Every object has a type.

object UUID
: A Universial Unique Identifier (UUID) that identifies a particular object. Both the Remote Procedure Call (RPC) runtime library and the Location Broker use object UUIDs to identify objects.

operation
: A procedure through which an object is accessed or manipulated.

port
: A specific communications endpoint within a host. A port is identified by a port number. *See also* **socket.**

port number
: One of the three parts in a socket address. For example, the character string *77* might represent a port number, while *ip:wooster[77]* might represent a socket address.

protocol family
: A set of communications protocols, for example, the Department of Defense Internetwork Protocols. All members of a protocol family use a common addressing mechanism to identify endpoints. The terms *protocol family* and *address family* are synonymous.

register an object and an interface with the Location Broker
: To enter in the Location Broker database an object and the location of a server that exports an interface for that object. Servers register with the Location Broker. Clients can use Location Broker lookup calls to determine the locations of registered objects.

register an object with the Remote Procedure Call (RPC) runtime library
: To enter an object and its location with the `rpc_$register_mgr` routine. An internal mechanism that allows an application to make remote procedure calls to remote hosts.

remote procedure call
: An invocation of a remote operation. You can make remote procedure calls between processes on different hosts or on the same host.

Remote Procedure Call (RPC) runtime library
: The set of `rpc_$` library routines that DECrpc provides to implement a remote procedure call mechanism.

server
: A process that implements interfaces. In the context of this manual, a server whose procedures can be invoked from remote hosts. A server exports one or more interfaces to one or more objects.

socket
: An endpoint of communications in the form of a message queue. A socket is identified by a socket address. *See also* **Berkeley UNIX socket abstraction.**

socket address
: A data structure that uniquely identifies a specific communications endpoint. A socket address consists of a port number and a network address.

type
: A class of object. All objects of a specific type can be accessed though the same interface or interfaces.

type UUID
: A Universal Unique Identifier (UUID) that identifies a particular type. Both the Remote Procedure Call (RPC) runtime library and the Location Broker use type UUIDs to specify types.

Universal Unique Identifier (UUID)
: An identifier used by the Network Computing System to identify interfaces, objects, and types.

well-known port
: A port whose port number is part of the definition of an interface. Clients of the interface always send to that port; servers always listen on that port. Some well-known ports are assigned to particular servers by the administrators of a protocol. For example, the administrators of the Internet Protocols have assigned the port number 23 to the `telnet` remote login facility.

Index

N

O

P

Q

R

S

T

U

V

W

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-234-1998 using a 1200- or 2400-baud modem from anywhere in the USA, Canada, or Puerto Rico. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital Subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal*	————	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

ULTRIX
Guide to the Location Broker
AA-PBKSA-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

What do you like best about this manual? _____

What do you like least about this manual? _____

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

What version of the software described by this manual are you using? _____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear – Fold Here and Tape

Do Not Tear – Fo

digital™

digit

BUSINESS REPLY MAIL

FIRST–CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3–2/Z04
110 SPIT BROOK ROAD
NASHUA NH 03062–9987

Do Not Tear – Fold Here

Do Not Tear –