HP DCE for OpenVMS Alpha and OpenVMS I64

Product Guide

Order Number: BA361-90002

January 2005

This guide provides an overview of the HP Distributed Computing Environment (DCE) for OpenVMS Alpha and OpenVMS Industry Standard 64 (I64) Version 3.2 and describes value-added features provided with HP DCE.

Revision/Update Information:	This guide supersedes the *Compaq DCE for OpenVMS VAX and OpenVMS Alpha Product Guide Version 3.0.*
Operating System:	OpenVMS Alpha Version 7.3-2 or higher OpenVMS I64 Version 8.2
Software Version:	HP DCE for OpenVMS Version 3.2

Hewlett-Packard Company
Palo Alto, California

Printed in the US

ZK6532

The HP OpenVMS documentation set is available on CD-ROM.

This document was prepared using VAX DOCUMENT, Version 2.1.

Contents

4 Using HP DCE with DECnet

5 Using HP DCE with Microsoft's NT LAN Manager (NTLM)

6 Directory Names, Filenames, and Locations Across DCE Platforms

7 Application Development Considerations and Differences

8 Integrated Login

13 Development of Distributed Applications with FORTRAN

14 Troubleshooting

Preface

The *HP DCE for OpenVMS Alpha and OpenVMS I64 Product Guide* provides users of the HP Distributed Computing Environment (DCE) with the following information:

- An overview of DCE and the contents of the HP DCE product.
- The differences between using DCE of OSF/1 and on OpenVMS systems.
- The value-added features provided with HP DCE for OpenVMS product.

Intended Audience

This guide is intended for:

- Experienced programmers who want to write client/server applications.
- Experienced programmers who want to port existing applications to DCE.
- System managers who manage the distributed computing environment.
- Users who want to run distributed applications.

Document Structure

This guide is organized as follows:

- Chapter 1 provides an overview of the HP DCE OpenVMS product and describes its contents, restrictions, and additional features.
- Chapter 2 describes the DCE system configuration utility.
- Chapter 3 describes interoperability and compatibility issues.
- Chapter 4 discusses using the kit with DECnet networks.
- Chapter 5 discusses authenticating RPC using Microsoft's NT LAN Manager.
- Chapter 6 provides information about the names of DCE files and directories.
- Chapter 7 provides information about developing applications on OpenVMS Alpha and OpenVMS I64 systems.
- Chapter 8 provides information about Integrated Login.
- Chapter 9 provides tips on intercell naming.
- Chapter 10 describes the Enhanced Browser for viewing the CDS namespace.
- Chapter 11 describes general enhancements to the IDL compiler.
- Chapter 12 describes debugging support for RPC applications.
- Chapter 13 contains information for developing distributed FORTRAN programs on OpenVMS systems.
- Chapter 14 provides checklists and troubleshooting hints.

- Chapter 15 describes the example programs supplied with the Application Developer's Kit.
- The appendix describes the namespace editor (NSedit), a system management tool.

Related Documents

For additional information about HP OpenVMS products and services, visit the following World Wide Web address:

`http://www.hp.com/go/openvms/`

For DCE specific documentation, visit the following World Wide Web address:

`http://h71000.www7.hp.com/DOC/dce32.html`

Reader's Comments

HP welcomes your comments on this manual. Please send comments to either of the following addresses:

Internet	**openvmsdoc@hp.com**
Postal Mail	Hewlett-Packard Company OSSG Documentation Group, ZKO3-4/U08 110 Spit Brook Rd. Nashua, NH 03062-2698

How To Order Additional Documentation

For information about how to order additional documentation, visit the following World Wide Web address:

`http://www.hp.com/go/openvms/doc/order/`

Conventions

VMScluster systems are now referred to as OpenVMS Cluster systems. Unless otherwise specified, references in this document to OpenVMS Clusters or clusters are synonymous with VMSclusters.

The following conventions are also used in this guide:

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *device-name* contains up to five alphanumeric characters).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.

`Monospace type`	Monospace type indicates code examples and interactive screen displays. In the C programming language, monospace type in text identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.
Case-sensitivity	OpenVMS operating system commands do not differentiate between uppercase and lowercase. However, many DCE commands do make this distinction. In particular, the system configuration utility interprets names in a case-sensitive manner.

1
Overview of HP DCE

Distributed computing services, as implemented in the HP Distributed Computing Environment (DCE), provide an important enabling software technology for the development of distributed applications. DCE makes the underlying network architecture transparent to application developers. It consists of a software layer between the operating system/network interface and the distributed application program. It provides a variety of common services needed for development of distributed applications, such as name and time services, and a standard remote procedure call interface.

HP DCE for OpenVMS Alpha and OpenVMS I64 provides a means for application developers to design, develop, and deploy distributed applications. This release supports both the OpenVMS Alpha and OpenVMS I64 systems.

1.1 Kit Contents

HP DCE for OpenVMS Alpha and OpenVMS I64 consists of the following distributed computing technologies:

- DCE Remote Procedure Call (RPC) — Allows you to create and run client /server applications.
- DCE Cell Directory Service (CDS) — Provides location-independent naming for servers.
- DCE Distributed Time Service (DTS) — Provides synchronization of time in distributed network environments.
- DCE Security Service — Provides secure communications and controlled access to resources.
- Threads — Provides user control and synchronization of multiple operations.
- Interface Definition Language (IDL) Compiler — Provides the compiler required for developing distributed DCE applications.

1.2 Contents of Each Kit

HP DCE for OpenVMS has four kits available:

- Runtime Services Kit
- Application Developer's Kit
- CDS Server Kit
- Security Server Kit

Note that the right to use the Runtime Services Kit is included as part of the OpenVMS license. The other kits each require a separate license. You must install a kit on each system that will use DCE services.

The following sections list the contents of each of these kits.

1.2.1 Runtime Services Kit

The Runtime Services provide the basic services required for DCE applications to function. The Runtime Services Kit contains the following:

- Authenticated CDS Advertiser and Client Support
- CDS Browser
- CDS Control Program (cdscp)
- Authenticated DCE RPC runtime support (supports DECnet, TCP/IP, and UDP)
- RTI (Remote Task Invocation) RPC for HP's ACMSxp TP product on OpenVMS Alpha
- Security Client Support
- Integrated Login
- A DCE_LOGIN tool for obtaining credentials
- A RGY_EDIT tool for registry maintenance functions
- KINIT, KLIST, and KDESTROY Kerberos tools
- An ACL_EDIT tool for access control lists (ACLs) for DCE objects
- RPC Control Program (rpccp)
- Native Kerberos
- XDS Directory Services
- XDS Object Management
- NTLM Security support
- New APIs which support impersonation
- DCE Control Program (dcecp)
- Authenticated RPC runtime support (supports DECnet, TCP/IP, and UDP using NTLM security protocol on OpenVMS Alpha Version 7.2-1 and higher.)
- LDAP client support for nsid and gda

1.2.2 Application Developer's Kit

The Application Developer's Kit is used by developers to build DCE applications. The Application Developer's Kit contains the following:

- The above contents of the Runtime Services Kit
- Required DCE application development header files
- Interface Definition Language (IDL) compiler
- Object-Oriented RPC
- Generic Security Service (GSSAPI)
- LSE Templates for IDL
- Include (.H) files and .IDL files for application development
- Sample DCE applications

- An aid for porting MS RPC applications to DCE RPC applications

1.2.3 CDS Server Kit

The CDS Server kit provides the naming services necessary for DCE clients to locate DCE server applications. The CDS Server kit includes the following:

- CDS server (cdsd)
- Global Directory Agent (GDA)

  The Global Directory Agent (GDA) lets you link multiple CDS namespaces using the Internet Domain Name System (DNS), X.500, or LDAP.
- Name Services Interface Daemon (nsid); also known as the PC Nameserver Proxy

1.2.4 Security Server Kit

The Security Server kit provides the security services necessary for authenticated RPC calls between DCE client and server applications to function. The kit includes the following:

- Security server (secd)
- Tool used to create the security database (sec_create_db)
- Security server administrative tool (sec_admin)
- Auditing support

1.3 Platforms and Network Transports Supported by HP DCE

HP DCE is supported on OpenVMS Alpha Version 7.3-2 or higher and on OpenVMS I64 Version 8.2.

This version of HP DCE provides RPC communications over the following network protocols:

- UDP/IP
- TCP/IP
- DECnet Phase IV or DECnet/OSI

DCE on OpenVMS allows the user to select specific network protocols rather than defaulting to any on the supported list. You may restrict DCE to one or more specific protocols by setting the systemwide logical name RPC_SUPPORTED_ PROTSEQS to a list of network protocols, delimited by colons. The following example restricts DCE to only TCP/IP and UDP/IP protocols (disabling DECnet):

```
$ define/system/exec RPC_SUPPROTED_PROTSEQS "ncacn_ip_tcp:ncadg_ip_udp"
```

1.4 Online Help and Online Manual Pages

HP DCE provides online help for both the management of DCE services and the development of distributed applications. This DCL help is organized to maintain the reference page categories established in the OSF DCE documentation and online reference pages. These categories are user commands (1), application development support (3), driver and networking support (7), and administrative support (8).

To access the DCE reference information, use the HELP command. You can get extensive help on the following DCE top-level topics:

```
DCE_CDS    DCE_DTS    DCE_IDL    DCE_INTRO
DCE_RPC    DCE_SECURITY          DCE_THREADS
```

For example, to get help on DTS, enter the following command:

```
$ HELP DCE DCE_DTS
```

1.5 Restrictions When Using HP DCE

HP DCE Version 3.2 for OpenVMS Alpha and OpenVMS I64 does not provide all the functions of the full OSF DCE. The following components are not included in this DCE product; however, the full OSF documentation is included.

- DCE Distributed File Service (DFS)
- DCE Diskless Services

1.6 Threads

The threads interface is an important part of the architecture for DCE, and the DCE services rely on it. POSIX Threads Library (formerly DECthreads) is provided as part of the OpenVMS Alpha and OpenVMS I64 operating systems.

Refer to the *Guide to DECthreads* in the HP OpenVMS operating system's documentation set for information about threads.

1.7 Using RPC Without CDS or Security

To use RPC only, you begin a configuration as follows:

```
$ @SYS$MANAGER:DCE$SETUP.COM CONFIGURE or
$ @SYS$MANAGER:DCE$RPC_STARTUP
```

The DCE Configuration Menu is displayed. From this menu, choose the RPC_ Only option. This option lets you use DCE RPC without a DCE cell. This option requires applications to use string bindings instead of the name service to find servers.

To communicate with an RPC server, an RPC client needs the server binding information. The server binding information includes the protocol sequences that the RPC server supports and the location (node name or node address) of the RPC server. When the RPC server is started, it registers its endpoints with the RPC daemon. It also exports the binding information to the name server if the name server exists. The RPC client then gets the binding information from the name server. When the name server is not available, the binding information must be provided to the RPC client through other mechanisms.

Users can incorporate in their RPC server code a mechanism for broadcasting the binding information on the network. However, this may not be a desired short-term solution. An easy workaround is for the users to pass the string binding to the RPC client and have the RPC client call the RPC routine to convert the string binding. In this case, the users who are running the RPC client need to know two things:

- Protocol sequences that the RPC server supports. For example: ncacn_ip_tcp, ncadg_ip_udp, and ncacn_dnet_nsp.

  Note that the RPC client may encounter a communication error if it picks a transport that is not supported by the server.

- Location of the RPC server (the node on which the RPC server runs). For example: DECnet node name, DECnet address, Internet address, and so on.

  You can get this information by executing the following commands on the server system after the RPC server is started.

  ```
  $ RUN SYS$SYSTEM:DCE$RPCCP
  RPCCP> SHOW MAPPING
  ```

See the Test1 example program for an example of using RPC without CDS and DCE Security servers.

1.8 Unsupported Network Interfaces

DCE on OpenVMS supports the user deselection of network interfaces on each system in a DCE cell. Use the logical RPC_UNSUPPORTED_NETIFS, which points to a list of network interfaces delimited by a colon (:) that you do not want to use.

RPC at initialization parses the list of network interfaces defined with the logical RPC_UNSUPPORTED_NETIFS, and builds a global list of network interfaces for deselection by RPC. The global list of network interfaces is parsed to ignore the deselected interfaces.

1.9 Supported Network Addresses

DCE on OpenVMS supports the user selection of network addresses on each system in a DCE cell. Use the logical RPC_SUPPORTED_NETADDRS to point to a list of network addresses delimited by a colon (:) that you want to use.

At initialization, RPC parses the list of network addresses defined with the logical RPC_SUPPORTED_NETADDRS, and builds a global list of network addresses for selection by RPC. The global list of network addresses is parsed to use only the selected addresses.

1.10 Impersonating a Client

DCE Version 3.2 allows a server to impersonate a client. This means that the server may run with the security credentials of the client. The capabilities of the client belong to the server. Table 1–1 lists the APIs that have been added to support this functionality.

Table 1–1 APIs Used to Impersonate a Server

API	Description
rpc_impersonate_client(binding_handle, *status)	Called by the server to act as a client application with the appropriate rights granted to the server.
rpc_revert_to_self(*status)	Called by the server to revert back to its original security context after impersonating a client.
rpc_revert_to_self_ex(binding_handle, *status)	Called by the server to revert back to its original security context after impersonating the client.

1.11 Features of HP DCE Not Included with the OSF DCE

HP DCE for OpenVMS Alpha and OpenVMS I64 provides the following value-added features to help users develop and deploy DCE applications:

- CDS Enhanced Browser
- IDL compiler enhancements
- RPC Event Logger utility
- Name service interface daemon (PC Nameserver Proxy Agent)
- DCL interfaces to DCE tools
- Integrated Login
- Object-Oriented RPC
- NTLM Security
- Impersonation

1.11.1 CDS Enhanced Browser

The CDS Enhanced Browser contains additional functions beyond those contained in the OSF DCE Browser. See the Enhanced Browser chapter for more information.

1.11.2 IDL Compiler Enhancements

The HP DCE IDL compiler includes the following features beyond those documented in the OSF DCE documentation:

- By default, the IDL compiler does not generate stub auxiliary (AUX) files, thus saving space.
- DCE IDL implements an extended array syntax.
- DCE IDL generates runtime routine templates.
- DCE IDL supports the use of HP Fortran.

See Chapters 12 to 15 for more information about IDL.

1.11.3 The RPC Event Logger Utility

HP provides the RPC Event Logger, which records information about operations relating to the execution of an application interface. See the chapter titled Application Debugging with the RPC Event Logger for details.

1.11.4 Name Service Interface Daemon (nsid) for Microsoft RPC

HP provides the name service interface daemon (nsid), also known as the PC Nameserver Proxy Agent, to allow RPC communication with personal computers running the DCE-compatible Microsoft RPC. The nsid enables an RPC application on MS–DOS, MS–DOS Windows, and Windows NT to perform name-service operations that are available through RPC, as if the RPC applications on the PC are directly involved in the full CDS namespace.

For more information on using PCs with DCE, refer to *Distributing Applications Across DCE and Windows NT* by Teague and Rosenberry.

Beginning with HP DCE for OpenVMS Version 3.0, you can use LDAP to access the name service interface daemon in addition to the previous communication methods. To use nsid with LDAP, you must configure the proper DCE environment using the DCE$SETUP.COM configuration program. Refer to the *HP DCE for OpenVMS Alpha and OpenVMS I64 Installation and Configuration Guide* for information on configuring DCE.

1.11.5 DCL Interfaces to DCE Tools

DCE is multiplatform software designed to be used and managed on many different operating systems. For that reason, HP has worked to keep as much of the standard OSF DCE interface available as possible within the OpenVMS environment. For example, you can define foreign commands to execute DCE tools and utilities as you do on a UNIX system.

Note that the OpenVMS operating system does not differentiate between commands using lowercase and uppercase characters, but operating systems based on UNIX are case-sensitive. Many of the standard DCE commands differentiate between lowercase and uppercase characters. Many literal strings that appear in text, examples, syntax descriptions, and function descriptions must be typed exactly as shown.

To assist users more accustomed to OpenVMS syntax and conventions, HP also provides DCL interfaces for the following DCE tools:

- IDL compiler
- Universal unique identifier generator (uuidgen) utility

Note that you can use these interfaces only on OpenVMS DCE systems; OSF DCE documentation includes no DCL interface information. For information about the available DCL interfaces, refer to the chapter on DCL command interfaces to DCE tools in the *HP DCE for OpenVMS Alpha and OpenVMS I64 Reference Guide*. Some of these interfaces can be enabled during installation and configuration.

1.11.6 Integrated Login

HP provides Integrated Login, which combines the DCE and OpenVMS login procedures. See Chapter 8 for more information.

1.11.7 Object-Oriented RPC

IDL has been extended to support a number of C++ language syntax features that provide a distributed object framework. The DCE RPC runtime environment now supports C++ bindings to remote objects. The combination of these new features creates an Object-Oriented RPC. (See Chapter 12 for more information.)

2
DCE System Configuration

HP DCE for OpenVMS Alpha and OpenVMS I64 includes a system configuration utility, SYS$MANAGER:DCE$SETUP.COM, that is used after the kit installation to configure and start the DCE services. The *HP DCE for OpenVMS Alpha and OpenVMS I64 Installation and Configuration Guide* provides important information about setting up your initial DCE environment. This chapter provides general information about the DCE configuration utility options and provides details about the clobber option.

HP recommends that you use only DCE$SETUP.COM, the DCE system configuration utility, to reconfigure and restart the HP DCE services. This utility ensures that the proper configuration and sequencing of DCE operations occur. For example, instead of starting the RPC daemon (dced) directly, use DCE$SETUP.COM to start and stop daemons.

The DCE system configuration utility invokes a number of other utilities while it is configuring and starting the DCE services and creates a log file called SYS$MANAGER:DCE$SETUP.LOG. This error log file can be helpful in diagnosing problems that may occur during the product installation or subsequent reconfiguration.

Note

In a VMScluster environment, you must configure each VMScluster node separately. Although a DCE kit can be installed clusterwide, DCE services need specific DECnet and/or TCP/IP addresses and endpoints for each host. You must configure each VMScluster node that will be part of a DCE cell. Configure the VMScluster nodes exactly as single nodes are configured.

2.1 Starting and Stopping the RPC Daemon

Starting from DCE Version 3.0 following enhancements have been made to the DCE System management command procedure DCE$SETUP.COM.

- dced, the RPC daemon, can be started with the new startup command procedure, DCE$RPC_STARTUP.COM.
- DCE$SETUP calls DCE$RPC_STARTUP to start dced instead of starting it directly.
- DCE$RPC_STARTUP does UCX parameter checking instead of DCE$SETUP.
- DCE$RPC_SHUTDOWN checks to see if any DCE components are running before it stops the dced.

- Invoking DCE$SETUP with the CLEAN or CLOBBER option no longer deletes the RPC endpoint database. However, the endpoints for any DCE components will be removed. To delete the entire RPC endpoint database, you must invoke DCE$RPC_SHUTDOWN with the CLEAN option.
- Modifications to informational messages have been made to remind the user of the above changes.

The RPC daemon can be started or stopped with the two new command files DCE$RPC_STARTUP.COM and DCE$RPC_SHUTDOWN.COM, which are located in SYS$COMMON:[SYSMGR].

To start the Remote Procedure Call daemon, complete the following:

1. Run DCE$RPC_STARTUP.COM.
2. Specify [NO]CONFIRM to turn user prompting on or off. CONFIRM is the default.

To stop the Remote Procedure Call daemon, complete the following:

1. Run DCE$RPC_SHUTDOWN.COM.
2. Specify the following options in any order:
   - [NO]CONFIRM to turn user prompting on or off. CONFIRM is the default.
   - CLEAN to delete all entries from the RPC endpoint database.

Note

The RPC daemon must not be stopped if any DCE components or RPC applications are running on the system.

2.2 Limiting RPC Transports

The RPC daemon can limit which protocols will be used by RPC applications. To restrict the protocols that can be used, set a logical name RPC_SUPPORTED_PROTSEQS that contains the valid protocols separated by a colon. Valid protocols are ncadg_ip_udp, ncacn_ip_tcp, ncacn_dnet_nsp.

To prevent RPC applications from registering endpoints that use UDP/IP, use the following command:

```
$ Define RPC_SUPPORTED_PROTSEQS "ncacn_ip_tcp:ncacn_dnet_nsp"
```

2.3 Using the DCE System Configuration Utility

To access the DCE system configuration utility menu, log in to the SYSTEM account and enter the following command:

```
$ @SYS$MANAGER:DCE$SETUP.COM
```

The system configuration utility displays the following menu:

```
1)  Config      Configure DCE services on this system
2)  Show        Show DCE configuration and active daemons
3)  Stop        Terminate all active DCE daemons
4)  Start       Start all DCE daemons
5)  Restart     Terminate and restart all DCE daemons
6)  Clean       Terminate all active DCE daemons and remove
                all temporary local DCE databases
7)  Clobber     Terminate all active DCE daemons and remove
                all permanent local DCE databases
8)  Test        Run Configuration Verification Program
0)  Exit        Exit this procedure
?)  Help        Display helpful information

Please enter your selection number:
```

To enter a system configuration menu command directly from the command line, type the following command:

```
$ @DCE$SETUP.COM command
```

where *command* is one of the system configuration commands described in Table 2–1.

Table 2–1 System Configuration Commands

Command	Description
config	The config command modifies the DCE configuration. To use this utility you must be logged in to either the SYSTEM account or an account with the same privileges as the SYSTEM account. The utility displays the current system configuration and then prompts for changes to the configuration. The default answers to the prompts depend on the existing configuration. To choose the default answer, press Return. You can also type a question mark (?) in response to any of the prompts to have help text displayed. A third choice is to enter new input at the prompt. After you select all the services, the utility displays the new configuration and asks whether the permanent configuration database should be updated. The utility optionally starts all of the daemons for the configured services and runs the Configuration Verification Program (CVP).
show	The show command displays the current DCE system configuration in read-only mode. You need WORLD privileges to execute this command. The *HP DCE for OpenVMS Alpha and OpenVMS I64 Installation and Configuration Guide* also provides information on this command.
stop	The stop command terminates all active DCE daemons. You must have the SYSPRV privilege to use this command.
start	The start command starts all DCE daemons based on the current DCE system configuration. You must have the SYSPRV privilege to use this command.
restart	The restart command terminates all active DCE daemons and restarts the daemons based on the current DCE system configuration. You must have the SYSPRV privilege to use this command.
clean	The clean command terminates all active DCE daemons. It deletes temporary local databases associated with DCE services on this system. You must have the SYSPRV privilege to execute this command. After you execute this command, you must restart the DCE services and applications. To restart the daemons after executing the clean command, use DCE$SETUP start.
clobber	The clobber command terminates all active DCE daemons. It deletes temporary and permanent local databases associated with DCE services on this system, including the DCE system configuration files and any portion of the RPC name service database for the cell that is maintained on this system. You must have the SYSPRV privilege to execute this command. After you execute this command, you must reconfigure the services on this system because clobber returns the system to the state it was in during the kit installation before the initial DCE system configuration was performed. To reconfigure the services and restart the daemons after executing the clobber command, use DCE$SETUP config.
test	The test command begins the Configuration Verification Program.
exit	The exit command allows you to exit from the DCE System Configuration menu without executing an option.

Implications of Using the clobber Command

Caution

The clobber command destroys a DCE cell. If you use it, you must reconfigure major portions of the cell. Using this command causes the following events:

- All temporary and permanent DCE databases and files are deleted, including:
  - Configuration databases:

    DCE$LOCAL:[000000]DCE_CF.DB (permanent database)

    DCE$LOCAL:[000000]DCE_SERVICES.DB (permanent database)

    Loss of these databases means you must reconfigure the host by entering @SYS$MANAGER:DCE$SETUP CONFIG.
- If the host on which the clobber command has been executed is the name service server for the cell, the namespace and all files are deleted.

  All name service entries and directories must be recreated. To recreate the DCE entries and directories by reconfiguring DCE on this host, you can enter the command @SYS$MANAGER:DCE$SETUP CONFIG. Users can create all user namespace entries and directories. You must restart the daemons either by responding YES at the configuration procedure's prompt, or by entering the command @SYS$MANAGER:DCE$SETUP START at a later time.

2.4 Kerberos

The DCE security server makes UDP port 88 (service name "kerberos5") available for use by native Kerberos clients for authentication.

Note

Kerberos realm names must match the cell name of the DCE security server.

Native kerberos5 clients have undergone minimal testing, and are currently unsupported. However, there are no known problems in this area. If this interoperability is important to your site, you may want to try it.

3
Interoperability and Compatibility

This chapter describes interoperability and compatibility issues for HP DCE for OpenVMS Alpha and OpenVMS I64. Information is provided on the following topics:

- Interoperability with other DCE systems
- Use with Microsoft RPC

  HP DCE for OpenVMS Alpha and OpenVMS I64 also interoperates with non-DCE systems that are running the Microsoft DCE-compatible RPC. This chapter briefly describes how to run the name service interface daemon (nsid), also known as the PC Nameserver Proxy Agent, which allows systems running the Microsoft RPC software to access a DCE name service.
- Use in VMScluster environments

  DCE and VMScluster environments are distributed computing systems. This chapter includes information to consider when using the two technologies together.

3.1 Interoperability with Other DCE Systems

HP DCE for OpenVMS Alpha and OpenVMS I64 provides RPC interoperability with HP's other DCE offerings, with several restrictions. HP DCE systems must have at least one network transport in common with a HP DCE client or server in order to communicate. For example, a HP DCE client system that supports only the DECnet transport cannot communicate with a DCE server that supports only the Internet transports (TCP/IP and UDP/IP).

This release provides RPC interoperability with other vendors' DCE offerings, with similar restrictions to those listed for other HP DCE offerings.

The Interface Definition Language provides a data type, (error_status_t), for communicating error status values in remote procedure calls. Data of the error_ status_t type is subject to translation to a corresponding native error code. For example, a "memory fault" error status value returned from a HP OSF/1 system to an OpenVMS system will be translated into the OpenVMS error status value "access violation".

In some cases, information is lost in this translation process. For example, an OpenVMS success or informational message is mapped to a generic success status value on other systems, because most non OpenVMS systems do not use the same mechanism for successful status values and would interpret the value as an error code.

3.2 Interoperability with Microsoft RPC

DCE systems can interoperate with non-DCE systems that are running Microsoft RPC. Microsoft supplies a DCE-compatible version of remote procedure call software for use on systems running MS–DOS, Windows, or Windows NT. Microsoft RPC systems can also use a DCE name service. The DCE name service can include the Cell Directory Service (CDS). Microsoft RPC servers can export and import binding information, and Microsoft RPC clients can import binding information. Thus, DCE servers can be located and used by Microsoft RPC clients and, similarly, Microsoft RPC servers can be located and used by DCE clients.

HP DCE for OpenVMS Alpha and OpenVMS I64 includes a name service interface daemon (nsid), also known as the PC Nameserver Proxy Agent, that performs DCE name service clerk functions on behalf of Microsoft RPC clients and servers. Microsoft RPC does not include a DCE name service. Microsoft RPC clients and servers locate an nsid using locally maintained nsid binding information. The binding information consists of the transport over which the nsid is available, the nsid's host network address, and, optionally, the endpoint on which the nsid waits for incoming calls from Microsoft RPC clients and servers. You must provide the nsid's transport and host network address (and, optionally, the nsid's endpoint) to Microsoft RPC clients and servers that want to use the DCE Directory Service with Microsoft RPC applications.

Note

Although your DCE cell may have several NSI daemons running, Microsoft RPC users need the binding for only one nsid. The nsid you choose must be running on a system that belongs to the same DCE cell as the DCE systems with which Microsoft RPC systems will communicate.

You can obtain the nsid binding information by running the rpccp show mapping command on the system where the nsid is running. The following example shows how to enter this command on an OpenVMS Alpha system where this release is installed. The nsid bindings are those with the annotation `NSID: PC Nameserver Proxy Agent V1.0`. Select the appropriate endpoint from among these bindings. In the following example, the nsid binding for the TCP/IP network transport is `ncacn_ip_tcp:16.20.16.141[4685]`.

```
$ rpccp
rpccp> show mapping

 mappings:
  .
  .
  .
   <OBJECT>          nil
   <INTERFACE ID>    D3FBB514-0E3B-11CB-8FAD-08002B1D29C3,1.0
   <STRING BINDING>  ncacn_ip_tcp:16.20.16.141[4685]
   <ANNOTATION>      NSID: PC Nameserver Proxy Agent V1.0

   <OBJECT>          nil
   <INTERFACE ID>    D3FBB514-0E3B-11CB-8FAD-08002B1D29C3,1.0
   <STRING BINDING>  ncacn_dnet_nsp:2.711[RPC03AB0001]
   <ANNOTATION>      NSID: PC Nameserver Proxy Agent V1.0
  .
  .
  .
```

For more information on using PCs with DCE, see *Distributing Applications Across DCE and Windows NT*.

3.3 Understanding and Using OSF DCE and VMScluster Technologies

This section describes the following:

- Similarities between VMScluster environments and DCE environments (cells)
- Differences between VMScluster environments and DCE environments (cells)
- Limitations on using DCE in a VMScluster environment
- Configuration issues when using DCE in a VMScluster environment

3.3.1 Similarities Between VMScluster Environments and DCE Cells

VMScluster technology as implemented by OpenVMS systems provides some of the same features of distributed computing that OSF DCE provides. Many of the VMScluster concepts apply to DCE, and it is easy to think of a VMScluster system as being a type of DCE cell.

The following attributes are shared by DCE and VMScluster environments:

- Both technologies create a multiple-system environment that you can view as an extended system. Multiple resources are pulled together to extend computing power beyond that of a single system and beyond that of systems with simple network connections.
- Both technologies use a common name to identify participants. In a VMScluster environment, you can identify all VMScluster members by the VMScluster alias name; in a DCE cell, you can identify all participants by a cell name.
- Both technologies use a common **namespace** to register name translations for resources that are shared across the extended system. Systems in a VMScluster environment use the OpenVMS logical name mechanism; a DCE cell uses the distributed directory service.
- Both technologies use a single account and password registry that is shared across the extended system. You can log in to a VMScluster alias name and gain access to VMScluster members and resources; you can log in to a DCE cell and gain access to DCE cell resources.
- Both technologies use a shared file system. When you log in to a VMScluster, you have access to all disks, directories, and files associated with the VMScluster environment. A full DCE implementation includes the Distributed File Service (DFS), which allows access to disks, directories, and files associated with the DCE cell. However, HP DCE for OpenVMS Alpha and OpenVMS I64 does not include DFS.

3.3.2 Differences Between VMScluster Environments and DCE Cells

VMScluster environments differ from DCE cells in two significant ways:

- The VMScluster alias implementation allows two network addresses for each VMScluster node.

- A VMScluster environment includes a connection-forwarding mechanism that allows an extended VMScluster environment to appear as a single entity on a network.

VMScluster environments support the concept of individual systems as nodes in the extended system. In DCE, individual systems are called hosts. In a VMScluster environment, each node effectively has two addresses: a network node address and the VMScluster alias address. These two addresses are used differently, as follows:

- You can use the VMScluster alias address to access shared resources, which are resources accessible from every node. Because each VMScluster node can access the resource, the VMScluster router simply forwards the connect request to a VMScluster node. Because it does not matter which node is used to access the resource, the connection is made automatically and transparently to the requester.
- Alternatively, you can access shared or non-shared resources by using the node address of a specific VMScluster node.

In DCE there is no such dual identity. All network addressing is done directly to a specified host. The DCE cell does not have a separate network address, and it does not perform any forwarding functions. To share resources across hosts, DCE applications can use replication (resource copies) or store the resources in the shared file system, DFS, if it is available.

The VMScluster environment connection-forwarding mechanism permits the entire extended system to appear on the network as a single addressable entity (the VMScluster alias address). Although DCE does not support a connection-forwarding mechanism, DCE can use the Remote Procedure Call (RPC) grouping mechanism to access shared resources in a distributed file system. This mechanism selects, from an available set, one host/server pair that provides access to the shared resource.

3.3.3 Limitations on Using DCE in a VMScluster System

DCE does not support VMScluster connection forwarding. DCE requires, instead, that all connection requests be made directly to a specific node in the VMScluster instead of to a VMScluster alias.

For example, if you start a DCE application server named whammy on VMScluster node HENDRX in a VMScluster named GUITAR (VMScluster alias name), binding information includes node HENDRX addressing information; it does not include VMScluster alias GUITAR addressing information. In turn, when a client wants to communicate with server whammy, it must retrieve binding information about the server. This binding information must contain address information for physical node HENDRX, not for the VMScluster alias GUITAR.

DCE makes use of VMScluster technology in the following ways:

- You can install DCE from a single kit installation for all nodes in the VMScluster environment. You can store all of the executables, libraries, message files, and other resources in the common VMScluster file system.
- You can use a single, shared account to execute the DCE daemon processes.

Although DCE installation and daemon processes are handled in a standard VMScluster manner, you must configure each VMScluster node individually to run DCE services. Some DCE services require node-specific information to be stored in the non-shared file system.

3.3.4 DCE and VMScluster Configuration Issues

Although DCE cells and VMScluster environments include exclusive lists of hosts (nodes), the boundaries of the two environments do not need to match each other. In a VMScluster environment, each node can be a member of only one extended cluster system. The same applies to DCE: each host is a member of only one cell. However, when you configure DCE and use it with VMScluster environments, the boundaries of a cell and the boundaries of a VMScluster do not need to be the same.

For security reasons, you should not have some members of a VMScluster belong to one cell and other members of a VMScluster belong to another cell. However, members of multiple VMScluster environments can be members of one DCE cell.

4

Using HP DCE with DECnet

The following sections describe information you need to know when using HP DCE for OpenVMS Alpha and OpenVMS I64 with DECnet software.

HP DCE for OpenVMS Alpha and OpenVMS I64 supports DECnet Phase IV networking. It also supports DECnet/OSI (DECnet Phase V).

4.1 DECnet and DCE Startup and Shutdown Sequences

Before you start or stop DECnet, you should first stop the DCE services. Then, after you start DECnet, restart the DCE services. Follow these steps to shut down DECnet and DCE on a system running DCE applications:

1. Stop any DCE applications that are running.
2. Stop the DCE services.

   If you are performing a system shutdown, DCE services are stopped with the following command, placed *before* the network transport shutdown commands in the site-specific shutdown procedure SYS$MANAGER:SYSHUTDOWN.COM:

   ```
   $ @SYS$STARTUP:DCE$SHUTDOWN
   ```

   This ensures that both DCE services *and* DECnet shut down in the correct order.

   If, however, you must shut down DECnet but are not performing a system shutdown, first stop the DCE services with this command:

   ```
   $ @SYS$MANAGER:DCE$SETUP clean
   ```

3. Then, if you are not performing a system shutdown, you can also stop DECnet interactively with one of the following commands:

   To shut down DECnet Phase IV, use the following command:

   ```
   $ MCR NCP SET EXECUTOR STATE SHUT
   ```

   To shut down DECnet/OSI, use the following command:

   ```
   $ @SYS$MANAGER:NET$SHUTDOWN
   ```

Here is the sequence to follow when you start DECnet on a system that is also running DCE applications:

1. Start DECnet Phase IV with the following command (usually executed from system startup procedures):

   ```
   $ @SYS$MANAGER:STARTNET.COM
   ```

   Start DECnet/OSI with the following command:

   ```
   $ @SYS$STARTUP:NET$STARTUP.COM
   ```

2. Make sure the DCE services are started.

Check to see that the DCE startup command procedure is invoked by the site-specific startup procedure. In SYS$MANAGER:SYSTARTUP_VMS.COM, make sure the following line was placed *after* the network transport startup commands:

```
$ @SYS$STARTUP:DCE$STARTUP.COM
```

DCE startup can occur only after successful completion of the DECnet startup procedure.

If you need to start the DCE services, but are not performing a system reboot, you can start DCE with this command:

```
$ @SYS$MANAGER:DCE$SETUP start
```

3. After the DCE services are started, you can restart your DCE applications.

4.2 Running DCE Server Applications Using DECnet

Users running server applications that support DECnet need to consider the following:

- The requirements needed on accounts running servers
- The restrictions on naming DECnet endpoints

4.2.1 Server Account Requirements

A DCE server application listening for client requests using the ncacn_dnet_nsp protocol sequence must be able to create a DECnet server endpoint (known as a named object in DECnet). To create the endpoint, the server application must run from an account that has either the rights identifier NET$DECLAREOBJECT or the privilege SYSNAM enabled.

If the NET$DECLAREOBJECT rights identifier does not already exist on your system, installation of HP DCE for OpenVMS Alpha and OpenVMS I64 creates it for you.

Use the OpenVMS Authorize utility (AUTHORIZE) to display the rights identifier, as follows:

```
$ RUN SYS$SYSTEM:AUTHORIZE

UAF> SHOW /IDENTIFIER NET$DECLAREOBJECT

    Name                            Value           Attributes
    NET$DECLAREOBJECT               %X91F50005      DYNAMIC
```

If a server application must run from an account without the SYSNAM privilege, and the rights identifier does not exist, you must use AUTHORIZE to grant the rights identifier to the account. For example:

```
$ RUN SYS$SYSTEM:AUTHORIZE
UAF> GRANT/IDENTIFIER NET$DECLAREOBJECT uic/account-specification
```

If the server account does not have the rights identifier NET$DECLAREOBJECT or the SYSNAM privilege, the RPC use-protocol-sequence API routines such as rpc_server_use_all_protseqs( ) and rpc_server_use_protseq( ) return the status code rpc_s_cant_listen_socket for the ncacn_dnet_nsp (DECnet) protocol sequence.

4.2.2 DECnet Endpoint Naming

To prevent RPC interoperability problems between DECnet–VAX and DECnet-UNIX hosts, HP recommends that you specify all well-known server endpoints completely in uppercase characters, using a maximum of 15 characters.

The following example shows an IDL file using an uppercase endpoint:

```
[uuid(43D2681B-A000-0000-0D00-00C663000000),
    version(1),
    endpoint("ncadg_ip_udp:[2001]",
       "ncacn_ip_tcp:[2001]",
             "ncacn_dnet_nsp:[APP_SERVER]")
]
interface my_app
```

When a server calls the RPC use-protocol-sequence API routines such as rpc_server_use_all_protseqs_ep( ) and rpc_server_use_protseq_if( ), DECnet on OpenVMS creates ncacn_dnet_nsp endpoints in uppercase characters, regardless of how the endpoint was specified. DECnet on OpenVMS also converts to uppercase the endpoints in all incoming and outgoing RPC requests.

DECnet-UNIX, however, does no conversions on ncacn_dnet_nsp endpoints. These differences can prevent client requests from reaching a server.

For example, an UNIX DCE server listening for client requests over the ncacn_ dnet_nsp protocol sequence with the endpoint app_server is not able to receive requests from an OpenVMS DCE client. Even though the OpenVMS client uses the endpoint app_server to create a binding handle (by using a string binding or from an import), DECnet on OpenVMS converts the endpoint in the outgoing RPC request to uppercase APP_SERVER. Because the UNIX DCE server application is listening on the lowercase app_server endpoint, the client request is rejected.

4.3 DECnet String Binding Formats Supported in This Release

To support the use of string bindings in this release, HP has added the following DECnet value to the list of supported protocol sequences:

ncacn_dnet_nsp

Unlike TCP/IP and UDP/IP, DECnet allows a named endpoint. An example of a DECnet protocol sequence named endpoint is TESTNAME. HP recommends that you use uppercase names with no more than 15 characters.

An example of an object number is #17. The # (number sign) character must precede an object number.

At present, there are no DECnet Phase IV options.

5

Using HP DCE with Microsoft's NT LAN Manager (NTLM)

Beginning with OpenVMS Alpha Version 7.2, RPC provides WINNT as an additional authentication service. WINNT, which is based on Microsoft's NTLM authentication protocol, allows you to build RPC client or server applications using WINNT authentication. These applications will allow secure communications in a Microsoft security environment.

5.1 Using WINNT Authentication with RPC Server Applications

To accept requests that use WINNT authentication, the RPC server application must do the following:

1. The server application must call rpc_server_register_auth_info( ) to tell the server RPC runtime that it supports WINNT authentication.
2. When the server application is run, it uses all WINNT security information from the current address space. If the process that deploys the server application has not acquired WINNT security information for its address space, then the RPC server's call to rpc_server_register_auth_info( ) will fail. To obtain WINNT security information, the NTA$LOGON utility must be run.

For an RPC server application to impersonate the requesting client, you must complete the following:

1. The first input parameter to each RPC server manager routine is a binding handle that represents the requesting client. If the RPC server application wants to impersonate the client represented by the binding handle, then the RPC server manager routine must call rpc_impersonate_client( ) with the binding handle as input. This allows the RPC server to use the WINNT and OpenVMS security information of the client instead of the WINNT and OpenVMS security information of the server.
2. When the RPC server application wants to run with its original WINNT and OpenVMS security information, it must call either rpc_revert_to_self( ) or rpc_revert_to_self_ex( ).

   Running with WINNT security information means that the RPC application accesses a resource on the system that has WINNT access control lists. The operating system checks the RPC application's WINNT security information to determine if access is allowed. If the application accesses a resource with OpenVMS ACLs, it is checked against the OpenVMS security information of the application.

For an RPC client application to send requests that will use WINNT authentication, you must complete the following:

1. The client application must call rpc_binding_set_auth_info( ) to set WINNT authentication information on the binding.
2. The client application must pass security credential information to rpc_ binding_set_auth_info( ). Use the rpc_binding_set_auth_info( ) *auth_ident* parameter value to pass WINNT security information. The *auth_ident* parameter can have one of the following two values:
   - The *auth_ident* parameter supplied to rpc_binding_set_auth_info( ) is NULL. If this is the case, then the client application will use all WINNT security information from the current address space. If the process that deploys the client RPC application has not acquired WINNT security information for it's address space, then the RPC client's call to rpc_ binding_set_auth_info( ) will fail. To obtain WINNT security information, the NTA$LOGON utility must be run.
   - Supply a valid rpc_auth_identity_handle_t obtained from a call to rpc_ winnt_set_auth_idenity( ). This option allows the client to use WINNT security information other than that of the current address space.

Note

Be careful when passing in the *auth_ident* parameter to perform authentication. If multiple invalid authentications occur, OpenVMS generates an intrusion record. Any subsequent valid authentications will fail. If this occurs, contact your system manager to delete the intrusion record.

5.2 RPC APIs Created or Enhanced to Support WINNT Authentication

The following routines have been created or enhanced to support the WINNT authentication service:

```
rpc_binding_set_auth_info()
rpc_server_register_auth_info()
rpc_binding_inq_auth_info()
rpc_binding_inq_auth_client()
rpc_mgmt_inq_dflt_authn_level()
rpc_mgmt_inq_server_princ_name()
rpc_winnt_set_auth_identity()
rpc_winnt_free_auth_identity()
rpc_impersonate_client()
rpc_revert_to_self()
rpc_revert_to_self_ex()
```

For more information on the RPC security APIs, see the *HP DCE for OpenVMS Alpha and OpenVMS I64 Reference Guide*.

5.3 Using the NTA$LOGON Utility

The NTA$LOGON utility allows client and server applications to obtain WINNT security information. This section provides NTLOGON syntax and usage examples. For more information on the NTA$LOGON utility, see the *HP COM, Registry, and Events for OpenVMS Developer's Guide.*

NAME

NTLOGON — Invokes the NTA$LOGON utility

SYNOPSIS

ntlogon *username password*

Note that all character strings will be converted to uppercase unless they are enclosed in double quotations (""").

QUALIFIERS

/LOG
Displays the result of a transaction.
/LIST
Lists the NT credentials for the current process. This is the natural persona.
/DELETE
Deletes the NT credentials for the current process.
/DOMAIN = *domain*
Specifies a different domain.

EXAMPLES

The following example shows how to use the NTA$LOGON utility:

```
$ ntlogon/list
[Persona #1 NT extension: Account= "TESTACCNT" Domain=
"OPENVMS_ARPC" ]

$ ntlogon/delete

$ ntlogon/list
ERROR: NtOpenProcessToken() failure: -1073741700
0xc000007c
%SYSTEM-E-NOSUCHEXT, no such extension found

$ ntlogon TESTSACCNT examplepassword

$ ntlogon/list
[Persona #1 NT extension: Account= "TESTACCNT" Domain=
"OPENVMS_ARPC" ]

$ ntlogon/log/domain=openvms_dcom "okelley" "password"
[Deleting existing NT extension]
[Persona #1 NT extension: Account= "okelley" Domain=
"OPENVMS_DCOM" ]
```

For more information on setting up your OpenVMS environment to use WINNT authentication, see the *HP COM, Registry, and Events for OpenVMS Developer's Guide.*

6

Directory Names, Filenames, and Locations Across DCE Platforms

This chapter provides the names and locations of important DCE directories and files as they are installed and used with HP DCE for OpenVMS Alpha and OpenVMS I64 systems. Tables show the correlation between HP DCE directories and files and their counterparts on other DCE kits.

6.1 DCE Directories

DCE installation and configuration creates a number of directories that are required for proper DCE execution. On HP DCE for OpenVMS Alpha and OpenVMS I64, you can access the top-level DCE directory by using the logical name DCE$LOCAL. This is the top-level DCE directory named DCE$LOCAL:[000000]. On a Tru64 UNIX system, the corresponding DCE local directory is created in /opt/dcelocal. The DCE services database, named dce_services.db, and the DCE configuration database, named dce_cf.db, reside in this top-level DCE local directory.

On HP DCE for OpenVMS Alpha and OpenVMS I64 systems, the DCE databases, which are created when the dced daemon starts, are located in the directory DCE$LOCAL:[VAR.DCED]. On a Tru64 UNIX system, these databases are located in the directory /opt/dcelocal/var/dced.

Table 6–1 lists the names of the DCE directories on HP DCE for OpenVMS Alpha and OpenVMS I64 and the corresponding directory names on HP DCE for Tru64 UNIX systems.

Table 6–1 DCE Directories for OpenVMS and Tru64 UNIX

OpenVMS DCE Directory Name	Tru64 UNIX Equivalent
DCE$LOCAL:[000000]	/opt/dcelocal
DCE$LOCAL:[VAR]	/opt/dcelocal/var
DCE$LOCAL:[VAR.DIRECTORY]	/opt/dcelocal/var/directory
DCE$LOCAL:[VAR.DCED]	/opt/dcelocal/var/dced

6.2 Setup Utilities

DCE installation also provides procedures and utilities to help you configure your DCE environment. On HP DCE for OpenVMS Alpha and OpenVMS I64, these procedures are placed in the SYS$MANAGER and SYS$STARTUP directories, with the exception of the DCE$DEFINE_OPTIONAL_COMMANDS.COM procedure, which is in the SYS$COMMON:[DCE$LIBRARY] directory. On a Tru64 UNIX system, equivalent utilities reside in /usr/sbin.

Table 6–2 lists the names of the HP DCE for OpenVMS setup command procedures and their equivalent Tru64 UNIX utilities.

Table 6–2 DCE Setup Utilities for OpenVMS and Tru64 UNIX

OpenVMS Filename	Tru64 UNIX Equivalent
DCE$DEFINE_OPTIONAL_COMMANDS.COM	NONE
DCE$DEFINE_REQUIRED_COMMANDS.COM	NONE
DCE$SETUP.COM	dcesetup
DCE$SHUTDOWN.COM	NONE
DCE$STARTUP.COM	NONE

6.3 Executable Images

Following installation on an OpenVMS Alpha or OpenVMS I64 system, all DCE executable images reside in the SYS$SYSTEM directory. On a Tru64 UNIX system, these images reside in /usr/bin.

Table 6–3 lists the names of the executable images on an OpenVMS system and the names of the equivalent images on a Tru64 UNIX system.

Table 6–3 Executable Images for OpenVMS and Tru64 UNIX

OpenVMS Filename	Tru64 UNIX Equivalent
DCE$ACL_EDIT.EXE	acl_edit
DCE$ADD_ID.EXE	NONE
DCE$AUDITD.EXE	auditd
DCE$CADUMP.EXE	cadump
DCE$CDSADVER.EXE	cdsadv
DCE$CDSBROWSER.EXE	cdsbrowser
DCE$CDSCLERK.EXE	cdsclerk
DCE$CDSCP.EXE	cdscp
DCE$CDSD.EXE	cdsd
DCE$CHPASS.EXE	NONE
DCE$CSRC	csrc
DCE$DCECP.EXE	dcecp
DCE$DCED.EXE	dced
DCE$DCE_LOGIN.EXE	dce_login
DCE$DCESX.EXE	dcesx
DCE$DTSCP.EXE	dtscp
DCE$DTSD.EXE	dtsd
DCE$DTS_NTP_PROVIDER	dts_ntp_provider
DCE$DTS_NULL_PROVIDER	dts_null_provider
DCE$EXPORT.EXE	NONE

(continued on next page)

Table 6–3 (Cont.) Executable Images for OpenVMS and Tru64 UNIX

OpenVMS Filename	Tru64 UNIX Equivalent
DCE$GDAD.EXE	gdad
DCE$GETCELLS.EXE	getcells
DCE$IDL.EXE	idl
DCE$IMPORT.EXE	NONE
DCE$KCFG.EXE	kcfg
DCE$KDESTROY.EXE	kdestroy
DCE$KINIT.EXE	kinit
DCE$KLIST.EXE	klist
DCE$LDAP_ADDCELL.EXE	ldap_addcell
DCE$LDAPDELETE.EXE	NONE
DCE$LDAPMODIFY.EXE	NONE
DCE$LDAPMODRDN.EXE	NONE
DCE$LDAPSEARCH.EXE	NONE
DCE$NSEDIT.EXE	NONE
DCE$NSID.EXE	nsid
DCE$RGY_EDIT.EXE	rgy_edit
DCE$RPCCP.EXE	rpccp
DCE$RPCLM.EXE	rpclm
DCE$SEC_ADMIN.EXE	sec_admin
DCE$SEC_CREATE_DB.EXE	sec_create_db
DCE$SECD.EXE	secd
DCE$SEC_SALVAGE_DB.EXE	sec_salvage_db
DCE$SEC_SETUP.EXE	NONE
DCE$SVCDUMPLOG.EXE	svcdumplog
DCE$TCL.EXE	NONE
DCE$UAF.EXE	NONE
DCE$UUIDGEN.EXE	uuidgen
DCE$X500_ADDCELL.EXE	x500_addcell

6.4 Library Images

Following installation on an OpenVMS Alpha or OpenVMS I64 system, all DCE library images reside in the SYS$LIBRARY directory. On a Tru64 UNIX system, these images reside in /usr/lib.

Table 6–4 lists the names of the library images on OpenVMS Alpha and OpenVMS I64 systems and the names of equivalent library images on an Tru64 UNIX system.

Table 6–4 DCE Library Images for OpenVMS and Tru64 UNIX

OpenVMS Filename	Tru64 UNIX Equivalent
DCE$IDL_CXX_SHR.EXE	NONE
DCE$KERNEL.EXE	NONE
DCE$LGI_CALLOUTS.EXE	NONE
DCE$LIB_SHR.EXE	libdce.a
DCE$SOCKSHR_IP.EXE	NONE
DCE$SOCKSHR_DNET_IV.EXE	NONE
DTSS$SHR.EXE	NONE
DTSS$RUNDOWN.EXE	NONE
DXD$CDS_SHR.EXE	NONE
DCE$NSEDIT_SHR.EXE	NONE
DCE$SOCKSHR_DNET_OSI.EXE	NONE
DCE$SOCKSHR_TPS.EXE	NONE
DCE$UAF_SHR.EXE	NONE
DCE$MSRPC_MAPPING_SHR.EXE	NONE

6.5 Message Files

After you install HP DCE for OpenVMS Alpha or OpenVMS I64, all DCE message files reside in the SYS$MESSAGE directory. On a Tru64 UNIX system, the message files reside in /usr/lib/nls/msg/en_US.88591.

Table 6–5 lists the names of the message files on an OpenVMS system and the names of equivalent files on a Tru64 UNIX system.

Table 6–5 Message Files for OpenVMS and Tru64 UNIX

OpenVMS Filename	Tru64 UNIX Equivalent
DCE$IDL_MSG.EXE	idl.cat
DCE$RPC_MSG.EXE	dcerpc.cat
DCE$UUIDGEN_MSG.EXE	uuidgen.cat
DCE$SEC_MSG.EXE	dcesec.cat
DCE$IL_MSG.EXE	NONE

6.6 Development Files

On an OpenVMS system, all DCE.h and .idl application development files reside in the SYS$COMMON:[DCE$LIBRARY] directory. You can also access this directory through the logical name DCE. On a Tru64 UNIX system, these files reside in the directory /usr/include/dce. Except for case-sensitivity differences between systems, all .h and .idl files have the same names on both OpenVMS and Tru64 UNIX systems.

6.7 Sample Applications

Both HP DCE for OpenVMS Alpha and OpenVMS I64 and HP DCE for Tru64 UNIX provide RPC and DCE sample applications. On OpenVMS, all example source and build files are located in the following separate subdirectories:

- SYS$COMMON:[SYSHLP.EXAMPLES.DCE]
- SYS$COMMON:[SYSHLP.EXAMPLES.DCE.RPC].

On Tru64 UNIX systems, the sample applications reside in subdirectories of /usr/examples/dce and /usr/examples/dce/rpc.

On both OpenVMS and Tru64 UNIX systems, example application files reside in subdirectories named for the sample applications. For example, on OpenVMS systems, all Distributed Calendar Program (book) example source and build files are located in the directory SYS$COMMON:[SYSHLP.EXAMPLES.DCE.RPC.BOOK].

On Tru64 UNIX systems, the equivalent files for the calendar program reside in the directory /usr/examples/dce/rpc/book.

7

Application Development Considerations and Differences

HP DCE for OpenVMS Alpha and OpenVMS I64 provides universal command interfaces, as well as directory structures, filenames, and application development environments that resemble those on UNIX systems. In general, this allows users to read any standard DCE documentation, such as that provided with this release, and create DCE applications on OpenVMS systems.

Although HP DCE for OpenVMS Alpha and OpenVMS I64 is designed to minimize differences from DCE as it is installed on UNIX systems, there are reasons to conform to OpenVMS standards and conventions first.

Primarily, users encounter the differences between the OpenVMS and UNIX platforms when they compile and link programs, but running compiled programs can require setup procedures specific to OpenVMS or this DCE kit.

This chapter describes application development formats and rules on OpenVMS systems that may differ from those described in the *OSF DCE Application Development Guide*. The following topics are discussed:

- Building applications
- Considering structure alignment with C compilers
- Building applications using OpenVMS object libraries
- Running applications
- Translating OSF DCE documentation examples to OpenVMS

7.1 Building Applications

This section describes command formats for compiling and linking applications on HP DCE for OpenVMS Alpha and OpenVMS I64. For general information about compiling and linking applications, refer to the *OSF DCE Application Development Guide*.

7.1.1 Linking DCE Applications

HP DCE uses the HP C Runtime library (HP - CRTL) to provide C runtime library functions for DCE software and DCE applications. DCE supports only the HP CRTL. HP DCE for OpenVMS Alpha and OpenVMS I64 has an options file, DCE:DCE.OPT which you should use for linking your DCE applications. This DCE.OPT options file includes SYS$SHARE:DCE$LIB_SHR and other libraries needed by DCE applications.

7.1.2 Considerations for Structure Alignment with C Compilers

On OpenVMS Alpha and I64 systems, DCE stub and library code assumes native, aligned form for structures. Do not use the C preprocessor pragma to prevent member alignment.

7.1.3 Considerations for Building DCE Applications Using OpenVMS Object Libraries

When moving programs from one operating system to another, you must consider the operations of different linkers. The following OpenVMS Linker operations are relevant to programmers developing DCE applications:

- The OpenVMS Linker does not load an object module from an object library unless the module is needed to resolve a reference in another component of the program.
- The OpenVMS Linker does not load an object module from an object library to resolve a reference to an external variable if that object module contains only a compile-time initialization of the variable. In this case, the Linker creates an uninitialized PSECT for the variable. References to this variable at program runtime will yield incorrect results.

These Linker operations are important to DCE application developers because the stub code produced by the IDL compiler contains only compile-time initialization for some external variables that will be referenced by DCE applications. To ensure that these variables are initialized properly, you must explicitly include the stub modules when you link your DCE application.

Suppose you are building the client portion of your DCE application, MYAPP. The MYAPP application contains two client stub modules, MYAPP_1_CSTUB.OBJ and MYAPP_2_CSTUB.OBJ, that are stored in an object library called MYLIB. To create the MYAPP executable code, enter the following link command:

```
$ LINK/EXE=MYAPP,MYLIB.OLB/LIB/INCLUDE=(MYAPP_1_CSTUB,MYAPP_2_CSTUB)
```

Use a similar linking method to create executable server code.

7.2 Running Applications

After you compile and link an application, the result is an executable image. For example, you may create an executable image named APPD.EXE.

If your application is a simple executable file, you can run it as you do any OpenVMS executable. However, if your application accepts command line switches or input that is unacceptable from DCL (such as -d), you must define a foreign command that can invoke the executable. For example, assign a symbol with a command such as the following:

```
$ APPD:== $WORK1:[CARL.MYDCETEST]APPD.EXE
```

This assignment allows you to run the application with a command such as the following:

```
$ appd -d
```

7.3 Translating OSF DCE Documentation Examples to OpenVMS

The *OSF DCE Application Development Guide* refers to files that do not exist on OpenVMS systems and illustrates commands and command syntax that do not work in an OpenVMS environment. The example in Section 7.1 includes a command line that illustrates many of the differences you see when you compile DCE code on OpenVMS. Note the following differences for writing applications on OpenVMS systems:

- The name for libdce is DCE$LIB_SHR.EXE.
- Object format files created by the IDL and C compilers have the file extension .OBJ instead of .o.
- On the HP Tru64 UNIX operating system, users typically define the environment variable **NLSPATH** to reference DCE message catalog files. On OpenVMS systems, there is no need for users to establish a symbolic reference to message catalog files.
- Using the UNIX command setenv is equivalent to defining an OpenVMS logical name. Note that many important logical names are defined during DCE configuration. See the installation and configuration guide for more information.
- OpenVMS does not have a program called MAKE or the makefiles that use it. On OpenVMS systems, you can convert these files to command procedures or use source-control software such as the HP/Module Management System (MMS) for OpenVMS.
- DECnet endpoints in IDL files must be uppercase and no longer than 15 characters. See the chapter on using HP DCE with DECnet for more information.
- The UNIX feature fork is generally analogous to spawning a subprocess in OpenVMS.
- Programs that accept arguments require foreign command assignment on OpenVMS. Unless you define a foreign command first, examples such as the following do not work:

  ```
  $ bookd -v
  $ test1 -d
  ```

- Any examples or descriptions that use specific UNIX commands or syntax do not work on OpenVMS systems. A command such as the following, which redirects output to a file called binop.idl, does not work on OpenVMS:

  ```
  $ uuidgen -i > binop.idl
  ```

  For this particular command, uuidgen, you can direct output to a file by using the command option -o as follows:

  ```
  $ uuidgen -i -o binop.idl
  ```

  You can substitute the OpenVMS equivalent SYS$DISK[ ] for the ./ syntax. However, to make an example such as this work on OpenVMS, you must first define a foreign command that points to the executable using standard OpenVMS directory syntax.

  In addition, the various UNIX commands for TYPE (cat), DIRECTORY (ls), SHOW PROCESS (ps), and STOP/ID=*process-id* (kill) do not work directly on OpenVMS systems.

- DCE filenames and locations are similar but different on OpenVMS. See the Directory Names, Filenames, and Locations Across DCE Platforms chapter for more information about the differences and similarities.

The PIPE command can be used to simulate some UNIX style redirection. Type `HELP PIPE` for more information. The PIPE command is only available on OpenVMS Version 7.0 or greater.

7.4 Mapping MSRPC Calls to DCE RPC Calls

The Microsoft RPC mapping file acts as a porting aid in mapping Microsoft RPC calls to DCE RPC calls. This mechanism is provided for OpenVMS Alpha Version 7.2 and higher and I64 Version 8.2 with the Application Developer's Kit.

To aid in porting Microsoft RPC applications to the DCE format, a new shareable image SYS$LIBRARY:MSRPC_MAPPING_SHR.EXE can be used to link with the RPC application. This new image provides entry points that map a subset of MSRPC calls to their DCE equivalents. To identify which APIs have been mapped, see the MSRPC_MAPPING.H file. This file needs to be included in the RPC application.

8
Integrated Login

This chapter discusses Integrated Login, a component of HP DCE for OpenVMS Alpha and OpenVMS I64 that combines the DCE and OpenVMS login procedures.

Integrated Login users should read the following sections:

- Section 8.1 — Overview
- Section 8.3 — Integrated Login Procedure
- Section 8.4 — Changing Your DCE Password
- Section 8.9 — Frequently Asked Questions for Users

System administrators should read the entire chapter (especially Section 8.5 and Section 8.10).

8.1 Overview

Integrated Login allows you to do the following:

- Obtain DCE credentials when you interactively log in to OpenVMS. There is no need for a separate DCE login.
- Enter either the DCE principal name and password or the OpenVMS username and password at the OpenVMS username and password prompt. (Using the DCE principal name and password is recommended.)
- Automatically synchronize DCE and OpenVMS passwords on every system throughout the cell that supports Integrated Login.
- Use local login if the DCE Security Service is unavailable.

Integrated login is different from single login. **Integrated login** means that the OpenVMS and DCE login processes are combined. When you log in to the OpenVMS system specifying a single username and password, you are automatically logged in to DCE as well. **Single login** means that once you have been authenticated on one system (that is, integrated login has occurred), you are automatically authenticated on any other system within the cell. (For example, with single login it would be possible for telnet not to prompt for a username and password.) DCE for OpenVMS Alpha and OpenVMS I64 provides integrated login, not single login.

Integrated Login occurs when you log in to a standard interactive session, start a remote interactive session, or create a terminal window. Integrated Login is not supported for network jobs, batch jobs, or detached processes.

8.2 Integrated Login Components

The components of Integrated Login include the following:

- Integrated Login procedure
- DCE User Authorization File (DCE$UAF)
- DCE$UAF Command Line Interface
- DCE IMPORT utility
- DCE EXPORT utility

8.3 Integrated Login Procedure

To log in to an OpenVMS system where DCE Integrated Login is enabled, perform the following steps:

1. At the OpenVMS username prompt, enter your DCE principal name or OpenVMS username.

   Note

   HP recommends that you specify your DCE principal name and password when logging in to a system on which Integrated Login is enabled.

   The DCE principal name you specify can contain no more than 32 characters. If your principal name and cell name combination contains more than 32 characters, specify the OpenVMS username that is associated with your DCE account instead. (This username is entered in the DCE$UAF file.) You should still enter your DCE password to obtain DCE credentials even if you specify your OpenVMS username.

   If the DCE principal name or cell name contains lowercase characters or OpenVMS special characters (for example, "/" and ","), enclose the entire entry in quotation marks.

   If a cell name is entered with a principal name, separate the two with an at sign (@). If you do not specify a cell name, the current DCE cell name is assumed. For example:

   ```
   Username: "JaneSmith@paper_cell.widget.com"
   ```

2. At the password prompt, enter your DCE password (recommended) or OpenVMS password. If you enter your DCE password and your OpenVMS password is not currently synchronized, Integrated Login attempts to reset your OpenVMS password to match your DCE password.

When you specify your principal name, Integrated Login maps the principal name to your OpenVMS username by performing a lookup in the DCE$UAF. Similarly, if you specify your OpenVMS username, Integrated Login maps the username to your principal name by performing a case-blind lookup in the DCE$UAF. (If the principal name or username you specify is not found in DCE$UAF, a regular OpenVMS login is attempted.)

When the lookup is complete, Integrated Login has obtained both your username and principal name. With that information, Integrated Login first attempts an OpenVMS login, then a DCE login. (The same password is used for both login attempts.) Depending on the principal name, username, and password you specify, four possible outcomes can occur, as follows:

- DCE Login: Success OpenVMS Login: Success

  If the password provided results in a valid OpenVMS login and a valid DCE login, the OpenVMS login succeeds and you have DCE credentials. Following is an example of a successful login:

```
        Welcome to OpenVMS (TM) Alpha Operating System, Version V7.2

Username: smith
Password:
   Welcome to OpenVMS Alpha (TM) Operating System, Version V6.1 on node NODE
    Last interactive login on Tuesday, 10-JAN-1995 08:25:33.67
    Last non-interactive login on Tuesday, 10-JAN-1995 08:18:52.81
%DCE-S-IL_DCECERT, Certified DCE login for SMITH as principal "/.../dce_cell.
widget.com/Smith"
$
```

- DCE Login: Failure OpenVMS Login: Success

  If the password provided results in a valid OpenVMS login but a failed DCE login, the OpenVMS login succeeds but you do not have DCE credentials. Integrated Login displays a message stating that the DCE login failed. For example:

```
        Welcome to OpenVMS (TM) Alpha Operating System, Version V7.2

Username: smith
Password:
   Welcome to OpenVMS Alpha (TM) Operating System, Version V6.1 on node NODE
    Last interactive login on Tuesday, 10-JAN-1995 08:26:49.50
    Last non-interactive login on Tuesday, 10-JAN-1995 08:18:52.81
%DCE-I-IL_VMSONLY, DCE login as principal "/.../dce_cell.widget.com/Smith"
failed, OpenVMS login to SMITH successful
$
```

  If the "DCE login required" feature is enabled, this outcome will fail and you will receive the user authorization failure message. See Section 8.5 for information on the DCE login required feature.

- DCE Login: Success OpenVMS Login: Failure

  If the password provided results in a valid DCE login but a failed OpenVMS login, you are given DCE credentials and you are logged in to OpenVMS. Integrated Login then attempts to set the OpenVMS password to match the DCE password. In the following example, the password was successfully synchronized:

```
        Welcome to OpenVMS (TM) Alpha Operating System, Version V7.2

Username: smith
Password:
   Welcome to OpenVMS Alpha (TM) Operating System, Version V6.1 on node NODE
    Last interactive login on Tuesday, 10-JAN-1995 08:13:00.47
    Last non-interactive login on Tuesday, 10-JAN-1995 08:18:52.81
%DCE-S-IL_DCECERT, Certified DCE login for SMITH as principal
"/.../dce_cell.widget.com/Smith"
%DCE-S-IL_VMSPWDSYNC, OpenVMS password synchronized with DCE password
$
```

  In this example, the password was not successfully synchronized:

```
        Welcome to OpenVMS (TM) Alpha Operating System, Version V7.2

Username: smith
Password:
   Welcome to OpenVMS Alpha (TM) Operating System, Version V6.1 on node NODE
    Last interactive login on Tuesday, 10-JAN-1995 06:44:16.15
    Last non-interactive login on Tuesday, 10-JAN-1995 07:02:58.95
%DCE-S-IL_DCECERT, Certified DCE login for SMITH as principal
"/.../dce_cell.widget.com/Smith"
%DCE-I-IL_ERRVMSPWD, Error synchronizing OpenVMS password with DCE password
-DCE-F-IL_INVPWDLEN, password length must be between 15 and 32 characters
$
```

- DCE Login: Failure OpenVMS Login: Failure

  If the password provided results in a failed DCE login and a failed OpenVMS login, you are not given DCE credentials and the OpenVMS login does not succeed. For example:

```
        Welcome to OpenVMS (TM) Alpha Operating System, Version V7.2

Username: smith
Password:
User authorization failure
```

  When a login fails, you do not receive a message stating the reason for the login failure. If you are a system administrator, you can enable auditing to see the reasons for login failures. To enable auditing, enter the following command:

```
$ SET AUDIT/ALARM/ENABLE=LOGFAIL=ALL
```

8.4 Changing Your DCE Password

HP recommends that you change only your DCE password. After changing your DCE password, the next time you log in to the OpenVMS system specifying your new DCE password at the OpenVMS password prompt, your OpenVMS password is changed to match your DCE password. There is no need to separately change your OpenVMS password.

To change your DCE password, invoke the **CHPASS** utility with an optional DCE principal name. For example, entering any of the following invokes the CHPASS utility:

```
$ chpass
$ chpass smith
$ mcr dce$chpass
$ mcr dce$chpass smith
```

If you do not specify a DCE principal name on the command line, the CHPASS utility obtains the DCE principal name from the current credentials. For example:

```
$ chpass
Old password:
New password:
Verification:
```

If the process does not have a default login context, you are prompted for your principal name. For example:

```
$ kdestroy
$ chpass
Please enter the principal name: smith
Old password:
New password:
Verification:
```

As you enter the old and new passwords, the terminal does not echo the input. Because echoing is turned off, the user is asked to enter the new password twice to verify the input.

SYS$COMMON:[SYSMGR]DCE$DEFINE_REQUIRED_COMMANDS.COM defines the DCE symbol CHPASS, which is used to invoke DCE$CHPASS. If this symbol is not defined in your environment, you can define the symbol as follows:

```
$ CHPASS :== $SYS$SYSTEM:DCE$CHPASS.EXE
```

8.5 Enabling/Disabling Integrated Login on Your OpenVMS System

By default, Integrated Login is not enabled on your system. To enable Integrated Login, go to the Configuration Modify menu, and select the following:

```
8)  Enable DCE Integrated Login
```

If Integrated Login is already enabled, the menu will display the following:

```
8)  Disable DCE Integrated Login
```

Select this option to turn off Integrated Login capabilities on your system.

Note

By enabling Integrated Login, you accept DCE password policies. This means that you may be reducing security on your OpenVMS system because the following OpenVMS password features are not available with Integrated Login enabled:

- Password history file
- Password dictionary
- Local site-specific password policies (for example, password expiration dates)

Each user on the OpenVMS system who wants to use Integrated Login must have an entry in the DCE$UAF file. DCE$UAF entries are created by using the DCE UAF utility (see Section 8.6) or by using the DCE IMPORT utility (see Section 8.7).

The DCE login required feature allows you to disable a user's account on all systems in the cell by simply removing that user's name from the DCE registry.

To enable the DCE login required flag, define the logical name DCE$IL_DCE_LOGIN_REQUIRED as follows:

```
$ DEFINE/SYSTEM/EXEC DCE$IL_DCE_LOGIN_REQUIRED TRUE
```

To disable the flag, enter the following command:

```
$ DEASSIGN/SYSTEM/EXEC DCE$IL_DCE_LOGIN_REQUIRED
```

8.5.1 Disabling a System Account for Integrated Login

When DCE is unavailable and Integrated Login is enabled with the DCE login required flag set, you are also prevented from logging in to OpenVMS. HP recommends that you do not include an entry for at least one system account in DCE$UAF. This disables that system account for Integrated Login, which ensures that you can log in to OpenVMS from that account even if DCE is unavailable.

8.5.2 Password Expiration Dates on User Accounts

This section contains information for system administrators who set up users' DCE and OpenVMS accounts.

If you use the password expiration date feature on accounts on your OpenVMS system, set the password expiration for the users' DCE and OpenVMS accounts to the same date (or set the OpenVMS expiration date to a slightly later date). In this case, if a user changes his DCE password when it expires, the next time the user logs in to OpenVMS, his OpenVMS password is updated.

If the DCE expiration date occurs first, or if the user does not update his DCE password when it expires, the user receives a message when he logs in stating that his OpenVMS password has expired. The user is forced to enter a new OpenVMS password if the DISFORCE_PWD_CHANGE flag is not set on the user's OpenVMS account. (By default, this flag is not set.) This is inconvenient and confusing for the user because the new OpenVMS password is not propagated back into the DCE registry. The next time the user logs in with the new OpenVMS password, he will be logged in to OpenVMS only, without DCE credentials.

8.5.3 Potential Integrated Login and SYSGEN Problems

The Integrated Login component of DCE uses the SYSGEN parameter LGI_CALLOUTS. LGI_CALLOUTS must be set to 1 only in the ACTIVE SYSGEN parameter set when DCE is running with Integrated Login enabled. LGI_CALLOUTS must never be set to 1 in the CURRENT SYSGEN parameter set — this would prevent all logins from occurring on a subsequent reboot of the system. The following paragraphs discuss the reasons for this restriction. See the Troubleshooting chapter for information on how to solve this problem if it occurs.

If Integrated Login is enabled on your system, the DCE startup and configuration procedure, DCE$SETUP.COM, sets the SYSGEN parameter LGI_CALLOUTS to 1 in the ACTIVE SYSGEN parameter set when DCE is started and resets the parameter when DCE is shut down. LGI_CALLOUTS must never be set to 1 in the CURRENT SYSGEN parameter set because, in that case, the next time the system is booted the LGI_CALLOUTS parameter is set in the ACTIVE SYSGEN parameter set *before* DCE is started. This prevents logins from occurring.

If the ACTIVE value of LGI_CALLOUTS is set to 1 when DCE and Integrated Login are not running, the following error is displayed when LOGINOUT attempts to run (for example, for interactive or batch logins):

```
No logical name match
```

Consequently, all users are prevented from logging in to the system.

This problem can occur if, for example, a SYSGEN parameter is modified in the following way while Integrated Login is enabled. This prevents logins because it causes LGI_CALLOUTS to be set to 1 the next time the system is booted.

```
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN> SET param value
SYSGEN> WRITE CURRENT
SYSGEN> EXIT
$
```

The correct way to modify a SYSGEN parameter is to make the change in MODPARAMS.DAT and then run AUTOGEN. If it is essential to modify a SYSGEN parameter without using MODPARAMS.DAT and AUTOGEN, you must ensure that if you use ACTIVE, you write the parameters into ACTIVE only; and if you use CURRENT, you write the parameters into CURRENT only. Do *not* copy the ACTIVE parameters into CURRENT.

Following are two examples of acceptable ways to modify a SYSGEN parameter:

```
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN> USE CURRENT
SYSGEN> SET param value
SYSGEN> WRITE CURRENT
SYSGEN> EXIT
$
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN> USE ACTIVE       ! optional, default is ACTIVE
SYSGEN> SET param value
SYSGEN> WRITE ACTIVE
SYSGEN> EXIT
$
```

8.6 DCE User Authorization File (DCE$UAF)

The DCE User Authorization File (DCE$UAF) contains DCE account information about users who have an OpenVMS account on the local system and who want to use Integrated Login. DCE$UAF maps an OpenVMS account name to a DCE principal name, and is a logical extension to the OpenVMS System User Authorization File (SYSUAF).

8.6.1 DCE$UAF File Information

The DCE UAF utility is shipped as an OpenVMS executable image named DCE$UAF.EXE. The image resides in the SYS$SYSTEM directory.

The DCE$UAF database is an OpenVMS file that by default is named DCE$UAF.DAT and resides in SYS$SYSTEM. You can change the name or location, or both, of this file by defining the logical name DCE$UAF to point to the new filename and location.

8.6.2 Running the DCE$UAF Utility

Integrated Login includes a command line interface to the DCE$UAF utility that allows system administrators to create, edit, and display DCE$UAF records. See the *HP DCE for OpenVMS Alpha and OpenVMS I64 Reference Guide* for detailed descriptions of the DCE$UAF commands.

Integrated Login provides two methods of running the DCE$UAF utility, as follows:

- By invoking the DCE$UAF utility using a predefined symbol.

  ```
  $ DCE$UAF
  DCEUAF>
  ```

You can also specify a single DCE$UAF command on the command line. Control returns to DCL after the command is executed.

```
$ DCE$UAF command
$
```

SYS$COMMON:[SYSMGR]DCE$DEFINE_REQUIRED_COMMANDS.COM defines the DCE symbol DCE$UAF, which is used to invoke the DCE$UAF utility. If this symbol is not defined in your environment, you can define the symbol as follows:

```
$ DCE$UAF :== $SYS$SYSTEM:DCE$UAF
```

- By issuing the RUN command.

```
$ RUN SYS$SYSTEM:DCE$UAF
DCEUAF>
```

8.7 DCE Registry Import

The DCE IMPORT utility allows you to create principal and account entries in a DCE registry based on accounts in an existing OpenVMS authorization file. It is used for the following purposes:

- To populate the DCE registry when a new DCE cell is first established
- To add entries to an existing DCE registry when a new OpenVMS system joins an existing DCE cell
- To add entries to an existing DCE registry when new users have joined an OpenVMS system that is already part of an existing DCE cell

The DCE IMPORT utility also creates and maintains an **exclude list**. The exclude list contains the OpenVMS usernames of users who do not have, and do not require, a DCE account. This feature allows DCE IMPORT to skip over these users during import operations.

Note

The DCE IMPORT utility described in this section cannot be satisfied by the import function shipped with OSF DCE because of substantial differences between OpenVMS and UNIX user registry data.

Passwords cannot be imported. Instead, the automatic synchronization feature that occurs during integrated login is used to import user passwords.

8.7.1 DCE IMPORT File Information

The DCE IMPORT utility is shipped as an OpenVMS executable image named DCE$IMPORT.EXE. The image resides in the SYS$SYSTEM directory.

The DCE IMPORT exclude file is named by default DCE$IMPORT_ EXCLUDE.DAT and also resides in SYS$SYSTEM. You can change the name or location, or both, of this file by defining the logical name DCE$IMPORT_ EXCLUDE to point to the new filename and location.

8.7.2 Running DCE IMPORT

The DCE IMPORT utility allows system administrators to create principal and account entries in a DCE registry based on accounts in SYSUAF.

Integrated Login provides two methods of running the DCE IMPORT utility, as follows:

- By invoking the DCE IMPORT utility using a predefined symbol.

  ```
  $ DCE$IMPORT
  IMPORT>
  ```

  You can also specify a single DCE IMPORT command on the command line. Control returns to DCL after the command is executed.

  ```
  $ DCE IMPORT command
  $
  ```

  SYS$COMMON:[SYSMGR]DCE$DEFINE_REQUIRED_COMMANDS.COM defines the DCE symbol DCE$IMPORT, which is used to invoke the DCE IMPORT utility. If this symbol is not defined in your environment, you can define the symbol as follows:

  ```
  $ DCE$IMPORT :== $SYS$SYSTEM:DCE$IMPORT
  ```

- By issuing the RUN command.

  ```
  $ RUN SYS$SYSTEM:DCE$IMPORT
  IMPORT>
  ```

See the *HP DCE for OpenVMS Alpha and OpenVMS I64 Reference Guide* for detailed descriptions of the DCE IMPORT commands.

8.8 DCE Registry Export

The DCE EXPORT utility allows you to create entries in an OpenVMS authorization file from an existing DCE registry.

Using the DCE EXPORT utility, you convert DCE registry entries (or a subset of the registry entries) into records in the OpenVMS SYSUAF file and rights database. Conversions are essentially a reversal of those made with the DCE IMPORT function.

Passwords cannot be exported. Instead, the automatic synchronization feature that occurs during integrated login is used to export user passwords.

The DCE EXPORT utility also creates and maintains an **exclude list**. The exclude list contains the DCE names of users who do not have, and do not require, an OpenVMS account. This feature allows DCE EXPORT to skip over these users during export operations.

Note

The DCE EXPORT utility described in this section cannot be satisfied by the export function shipped with OSF DCE because of substantial differences between OpenVMS and UNIX user registry data.

8.8.1 DCE EXPORT File Information

The DCE EXPORT utility is shipped as an OpenVMS executable image named DCE$EXPORT.EXE. The image resides in the SYS$SYSTEM directory.

The DCE EXPORT exclude file is named by default DCE$EXPORT_EXCLUDE.DAT and also resides in SYS$SYSTEM. You can change the name or location, or both, of this file by defining the logical name DCE$EXPORT_EXCLUDE to point to the new filename and location.

8.8.2 Running DCE EXPORT

The DCE EXPORT utility allows system administrators to create an OpenVMS authorization file from an existing DCE registry.

Integrated Login provides two methods of running the DCE EXPORT utility, as follows:

- By invoking the DCE EXPORT utility using a predefined symbol.

  ```
  $ DCE$EXPORT
  EXPORT>
  ```

  You can also specify a single DCE EXPORT command on the command line. Control returns to DCL after the command is executed.

  ```
  $ DCE EXPORT command
  $
  ```

  SYS$COMMON:[SYSMGR]DCE$DEFINE_REQUIRED_COMMANDS.COM defines the DCE symbol DCE$EXPORT, which is used to invoke the DCE EXPORT utility. If this symbol is not defined in your environment, you can define the symbol as follows:

  ```
  $ DCE$EXPORT :== $SYS$SYSTEM:DCE$EXPORT
  ```

- By issuing the RUN command.

  ```
  $ RUN SYS$SYSTEM:DCE$EXPORT
  EXPORT>
  ```

See the *HP DCE for OpenVMS Alpha and OpenVMS I64 Reference Guide* for detailed descriptions of the DCE EXPORT commands.

8.9 Frequently Asked Questions for Users

Q: What exactly does Integrated Login do for me?
A: It performs a DCE_LOGIN on your behalf when you interactively log in to an OpenVMS system. (You will see an informational message stating that the login was successful if the DCE_LOGIN occurs.)

Q: Are there any other benefits to using Integrated Login?
A: Yes. It allows you to use a single username and password across multiple systems and/or OpenVMS clusters. With Integrated Login, you can use the same account information to log in to your OpenVMS systems as you do to log in to your non OpenVMS systems.

Q: At the OpenVMS username prompt, do I enter my OpenVMS username or my DCE account (principal) name?
A: Either the username or principal name is valid.

Q: Which password should I use to log in to the OpenVMS system (my DCE password or my OpenVMS password)?
A: Your OpenVMS and DCE passwords are normally the same because OpenVMS attempts to synchronize your passwords. If your passwords are not the same, you should log in using your DCE password. This will cause your OpenVMS password to be set to the same value as your DCE password. You can log in with your OpenVMS password, but if you do so, your passwords will not be synchronized and you will not obtain DCE credentials.

Q: If I enter my OpenVMS username, can I then enter my DCE password (and vice versa)?
A: Yes. But remember that you will only get DCE credentials if you enter your DCE password.

Q: Is the input at the OpenVMS username case-sensitive?
A: Yes. And since this input is parsed by the standard DCL parsing routines, all text not enclosed in quotation marks is converted to uppercase. Therefore, if you want to enter a principal name of "Smith" you must enclose the text in quotation marks.

Q: My DCE password contains lowercase characters. Do I need to enclose my password in quotes?
A: No. The password is not parsed by the DCE parsing routines, so quotes are not needed.

Q: How do I keep my DCE and OpenVMS passwords in sync?
A: OpenVMS does this for you. Your password is automatically propagated from the DCE registry to the OpenVMS System User Authorization file (SYSUAF) when you log in to the OpenVMS system using your valid DCE password.

Q: Do OpenVMS passwords get copied to the DCE registry?
A: No. This is why Integrated Login users should always use their DCE password when logging in to an OpenVMS system. This way DCE and OpenVMS passwords will stay synchronized.

Q: How should I change my password?
A: You should use the CHPASS utility on any node in the cell. This will change your password in the DCE registry, and the next time you log in to an OpenVMS system (using the new password) your local OpenVMS password will be automatically updated.

Q: What if I update my password using the OpenVMS command SET PASSWORD?
A: Your password will only be changed on that OpenVMS system; it will not be updated in the DCE registry. The next time you log in to that system, if you use the new OpenVMS password you will receive an "OpenVMS only" login. If you use your old DCE password you will receive an Integrated Login and your password on the OpenVMS system will be resynchronized to your old DCE password.

Q: Will account passwords on the OpenVMS system stay synchronized through the password synchronization mechanism when the password is changed on a UNIX system?
A: Yes. A password is automatically propagated from the DCE registry to the OpenVMS System User Authorization file (SYSUAF) when a user logs in to the OpenVMS system. Note that this assumes that the UNIX system updates the user's password in the DCE registry, and not just on the local UNIX system.

Q: Can I use Integrated Login when I start a DECwindows session?
A: Yes.

Q: Which password do I enter to unpause my workstation?
A: You must always enter your current OpenVMS password to resume a paused DECwindows session (this is usually your DCE password since OpenVMS attempts to keep them synchronized).

8.10 Frequently Asked Questions for System Administrators

Q: How do I enable Integrated Login on my system?
A: Use the DCE setup utility. (See the *HP DCE for OpenVMS Alpha and OpenVMS I64 Installation and Configuration Guide* for more information.)

Q: Is Integrated Login enabled by default?
A: No. After you install HP DCE for OpenVMS Alpha and OpenVMS I64 Version 3.2, Integrated Login is initially disabled.

Q: I've enabled Integrated Login on my system by using the DCE setup utility, but it still does not work. Why not?
A: Integrated Login is only available to users who have an entry in the DCE Integrated Login authorization file (DCE$UAF). You must populate the DCE$UAF file before Integrated Login can be used. If a user does not have an entry in the DCE$UAF file, then he or she cannot use Integrated Login.

Q: What is the purpose of the DCE$UAF file?
A: Entries in this file associate OpenVMS account names with DCE account names.

Q: How do I populate the DCE$UAF file?
A: The *HP DCE for OpenVMS Alpha and OpenVMS I64 Reference Guide* provides full details. Essentially, you issue ADD commands similar to the following to get entries into the DCE$UAF file:

```
$ dce$uaf
DCEUAF> ADD SMITH "john"
```

This creates an entry for the OpenVMS account name "SMITH" and associates it with the DCE account name "john".

Q: All of my users have DCE account names that are similar to their OpenVMS account names (for example, "SMITH" on OpenVMS and "smith" on DCE). Do I need to enter the principal name in this case?
A: No. To make adding these entries easier, the ADD command defaults the principal name to the lowercase equivalent of the OpenVMS username if you do not specify the principal name. If your OpenVMS account name is "JONES" and your DCE account name is "jones" you can simply enter:

```
DCEUAF> ADD JONES
```

Q: Is there an easier way to populate the DCE$UAF file without typing each name?
A: If all or most of your account names are the same on DCE as they are on OpenVMS (except for the case), you can use the ADD/ALL command. This will create an entry in the DCE$UAF file for every record in the SYSUAF file, as follows:

```
DCEUAF> ADD/ALL
```

Q: Should every account be set up for Integrated Login?
A: HP does *not* recommend that you enable the SYSTEM account for Integrated Login. If you have problems with your DCE configuration, you should have an account that you can log in to where an integrated login is not attempted. Operator and field service accounts are other accounts that you might want to omit from Integrated Login.

Q: Will existing users who already have DCE accounts, but do not have OpenVMS accounts, be able to log in to the OpenVMS system?
A: No. For a user to be able to log in to an OpenVMS system, he must have an OpenVMS account in the SYSUAF file.

Q: What happens when a user who doesn't have an entry in the DCE$UAF file tries to log in to the OpenVMS system?
A: If the user specifies a valid OpenVMS username and password, then he will be logged in as usual (as if Integrated Login was not installed or enabled). If the user specifies a DCE account name, the login will fail.

Q: How can I create accounts in the DCE registry based on the contents of my existing system user authorization file (SYSUAF)?
A: The DCE IMPORT utility performs this task. See the *HP DCE for OpenVMS Alpha and OpenVMS I64 Reference Guide* for more information.

Q: How can I create accounts in the OpenVMS authorization file (SYSUAF) based on the contents of the existing DCE registry?
A: The DCE EXPORT utility performs this task. See the *HP DCE for OpenVMS Alpha and OpenVMS I64 Reference Guide* for more information.

8.11 Potential Integrated Login and OpenVMS External Authentication Problems

DCE Integrated Login is currently incompatible with OpenVMS External Authentication. Only one of the two methods for authenticating users with external (to OpenVMS) mechanisms can be used at any one time.

Although the DCE configuration program checks to see if the local system is set up to use External Authentication, the system may experience a conflict due to operator error. If this occurs, the DCE LGI_CALLOUTS will override the OpenVMS External Authentication, disabling the External Authentication and allowing Integrated Login to function normally. Any applications that depend on External Authentication may be adversely affected.

For more information on OpenVMS External Authentication, see the OpenVMS Operating System documentation.

9
Intercell Naming

This chapter provides tips for choosing a cell name and for managing cell names in the Domain Name System (DNS), LDAP, and in X.500. Additional details can be found in the chapter about global and cell considerations in the *OSF DCE Administration Guide — Introduction*.

The following are simple guidelines for naming cells:

- Do not configure a cell with the same name as another cell on the same network.
- Choose your cell name carefully.

The last item is especially important, because the naming formats for DNS and LDAP/X.500 are incompatible, and DCE does not currently support changing the name of a cell. Therefore, you must understand which method you are using for intercell communications before you name the cell.

9.1 Intercell Naming with DNS

Names in DNS are associated with one or more data structures called resource records. The resource records define cells and are stored in a data file, called `/etc/namedb/hosts.db`. The data file is used by the BIND name daemon (`named`). To create a cell entry, you must edit the data file and create two or more (if replicas) resource records for each CDS server that maintains a replica of the cell namespace root. Do not configure a cell with the same name as another cell on the same network.

9.1.1 Intercell Naming Example — DNS

The following examples show the steps you should take to set up intercell naming between two cells called `laser-cell.zko.dec.com` and `ruby-cell.zko.dec.com`. (A summary of this process is provided at the end of this chapter.) The two cells belong to the same BIND domain `zko.dec.com`. Host `laser.zko.dec.com` is the master CDS server for the `laser-cell.zko.dec.com` cell. Host `ruby.zko.dec.com` is the master CDS server for the `ruby-cell.zko.dec.com cell`.

The BIND server must be authoritative for the domains of both the cell name and the hostnames. The BIND master server requires the following entries in its `/etc/namedb/hosts.db` file:

```
laser.zko.dec.com. IN A 25.0.0.127
```

```
laser-cell.zko.dec.com. IN MX 1 laser.zko.dec.com.
laser-cell.zko.dec.com. IN TXT "1
130f1c81-4876-11cc-931d-08002b33f531
Master /.../laser-cell.zko.dec.com/laser_ch
124ded80-4876-11cc-931d-08002b33f531
laser.zko.dec.com"
ruby.zko.dec.com. IN A 25.0.0.149
ruby-cell.zko.dec.com. IN MX 1 ruby.zko.dec.com.
ruby-cell.zko.dec.com. IN TXT "1
c8f5f807-487c-11cc-b499-08002b32b0ee
Master /.../ruby-cell.zko.dec.com/ruby_ch

c84946a6-487c-11cc-b499-08002b32b0ee
ruby.zko.dec.com"
```

Note

The TXT records *must span only one line.* You need to do whatever is required with your text editor of choice to ensure this. Widening your window helps. You should also ensure that the quotes are placed correctly, and that the hostname is at the end of the record.

The information to the right of the TXT column in the Hesiod Text Entry (that is, 1 130f1c81-48...) comes directly from the `cdscp show cell /.:/ as dns` command. For example, to obtain the information that goes in the `laser.zko.dec.com` text record (TXT), you would go to a host in the `laser` cell, and enter the `cdscp show cell /.:/ as dns` command. Then, when the system displays the requested information, you would cut and paste this information into the record. This method ensures that you do not have any typing errors. If the cell contains one or more replicas, add the additional text record(s) in the same manner. Make sure cell names and hostname text in the record are identical for Master and Read-Only TXT record(s). Only the clearinghouse (**x-cell/x_ch**) and UUID values change.

On UNIX master bind server systems, ensure that the records that you have entered are valid by issuing a `kill -1 named-process-id` command. For OpenVMS systems, see the TCP/IP product-specific implementation documentation for equivalent functionality. This causes the `named` daemon to read in the new `hosts.db` file.

Your host must access the bind server for the intercell information. To accomplish this, set name service parameters for your particular TCP/IP. This causes cell names to be sent to and resolved by the bind server and not your "localhost". Check the TCP/IP product specific documentation for instructions on setting the name service as well as invoking the `nslookup` command to obtain the host address:

```
laser.zko.dec.com> nslookup

Default Server:  localhost
Address:  127.0.0.1, 25.0.0.32
```

Next, enter the names of the cells, as shown:

```
> set type=any

> ruby-cell.zko.dec.com

Server:  localhost
Address:  127.0.0.1

ruby-cell.zko.dec.com     text = "1 c8f5f807-487c-11cc-b499-08002b32b0ee
Master /.../ruby-cell.zko.dec.com/ruby-cell.zko.dec.com/ruby_ch
c84946a6-487c-11cc-b499-08002b32b0ee
ruby.zko.dec.com"
ruby-cell.zko.dec.com     preference = 1, mail exchanger = ruby.zko.dec.com

ruby.zko.dec.com inet address = 25.00.127
```

View the information and ensure that it is complete and correct.

Now that you have set up BIND, you must use the Security Service `rgy-edit cell` command to create a cross-cell authentication account in the local and foreign cells. This account allows local principals to access objects in the foreign cell as authenticated users and vice versa.

In the `laser-cell.zko.dec.com` cell, you must use the `rgy-edit cell` command to create an account for `/.../ruby-cell.zko.dec.com`. Refer to the Security Service commands in the *OSF DCE Administration Reference* for details on the `cell` command. After adding the account for `/.../ruby-cell.zko.dec.com` in the `laser-cell.zko.dec.com` cell, you should have an account entry that looks like the following:

```
krbtgt/ruby.zko.dec.com [none none]:*:101:12::/::
```

Note that the cell name is stripped of the path qualifier and is prefixed with `krbtgt`. The resulting name is used as the primary name for the cross-cell authentication account. You should now also have a principal entry that looks like the following:

```
krbtgt/ruby.zko.dec.com                  101
```

If a cell is reconfigured, changing its namespace and clearinghouse UUIDs, the `krbtgt` principal created by the cell command must be deleted using `rgy_edit` in the foreign cell. Note that for HP DCE for OpenVMS Alpha and OpenVMS I64 Version 3.2, the `krbtgt` principal must be deleted on both cells before the cell command is reexecuted between two cells. To test for proper configuration, show the cell information for the foreign cell. For example, in the `laser-cell.zko.dec.com` cell, use the `cdscp show cell` command to show information about the `ruby-cell.zko.dec.com` cell. To do this at a `laser` cell host, execute the following command:

```
cdscp> show cell  /.../ruby-cell.zko.dec.com
```

To perform a similar operation from a `ruby-cell` cell host, execute the following command:

```
cdscp> show cell  /.../laser-cell.zko.dec.com
```

9.2 Intercell Naming with X.500

The DCE configuration program automatically creates an entry in the X.500 namespace for the cell when it is configured if the following conditions are true:

- The parent entry already exists.
- The cell name entry is not in use.

9.2.1 Intercell Naming Example — X.500

The following examples show the steps you should take to set up intercell naming between two cells called /c=us/o=hp/ou=lasercell and /c=us/o=hp/ou=rubycell. (A summary of this process is provided at the end of this chapter.) The two cells belong to the same X.500 namespace /c=us/o=hp. Host laser is the CDS master server for the /c=us/o=hp/ou=lasercell cell. Host ruby is the CDS master server for the /c=us/o=hp/ou=rubycell cell.

Note

X.500 cell names can contain spaces or hyphens if they are enclosed in double quotes, but underscores are never allowed, even if they are enclosed in double quotes. For example, the X.500 cell names /c=us/o=hp /ou="excess cell" and /c=us/o=hp/ou="excess-cell are allowed, but /c=us/o=hp/ou=excess_cell and /c=us/o=hp/ou="excess_cell are not allowed.

Answer "Yes" to the question "Do you want to register the DCE cell in X.500" during configuration of the cell. This puts the required DCE CDS information into the X.500 namespace for later use by GDA. This operation requires an X.500 DUA on the host system. Refer to *HP X.500 Directory Service — Management* for more information about installing and configuring X.500.

Execute an intercell command similar to the following command to show the root of the new cell and to see if everything works:

```
cdscp> show cell /.../c=us/o=hp/ou=rubycell
```

Enter the preceding command from an unauthenticated, nonprivileged account.

Now that you have configured and set up X.500, you must use the Security Service rgy_edit cell command to create a cross-cell authentication account in the local and foreign cells. This account allows local principals to access objects in the foreign cell as authenticated users and vice versa.

In the /c=us/o=hp/ou=lasercell cell, you must use the rgy_edit cell command to create an account for /.../c=us/o=hp/ou=rubycell. (Refer to the Security Service commands in the *OSF DCE Administration Reference* for details about the cell command.) After adding the account for /.../c=us/o=hp/ou=rubycell in the /c=us/o=hp/ou=lasercell cell, you should have an account entry that looks similar to the following:

```
krbtgt/c=us/o=hp/ou=rubycell [none none]:*:101:12::::
```

Note that the cell name is stripped of the path qualifier and is prefixed with krbtgt. The resulting name is used as the primary name for the cross-cell authentication account. You should now also have a principal entry that looks like the following:

```
krbtgt/c=us/o=hp/ou=rubycell                    101
```

If a cell is reconfigured, changing its namespace and clearinghouse UUIDs, the krbtgt principal created by the cell command must be deleted using rgy_edit in the foreign cell. Note that for HP DCE for OpenVMS Alpha and OpenVMS I64 Version 3.2, the krbtgt principal must be deleted on both cells before the cell command is reexecuted between two cells. To test for proper configuration, show the cell information for the foreign cell. For example, in the /c=us/o=hp /ou=lasercell cell, use the cdscp show cell command to show information about

the /c=us/o=hp/ou=rubycell cell. To do this at a laser cell host, execute the following command:

```
cdscp>  show cell /.../c=us/o=hp/ou=rubycell
```

To perform a similar operation from a /c=us/o=hp/ou=rubycell cell host, execute the following command:

```
cdscp>  show cell /.../c=us/o=hp/ou=lasercell
```

9.3 Intercell Naming with LDAP

The DCE configuration program automatically creates an entry in the LDAP namespace for the cell when the following conditions are true:

- The parent entry already exists.
- The cell name entry is not in use.

9.3.1 Intercell Naming Example — LDAP

The following examples show the steps you should take to set up intercell naming between two cells called /c=us/o=hp/ou=lasercell and /c=us/o=hp/ou=rubycell. (A summary of this process is provided at the end of this chapter.) The two cells belong to the same LDAP namespace /c=us/o=hp. Host laser is the CDS master server for the /c=us/o=hp/ou=lasercell cell. Host ruby is the CDS master server for the /c=us/o=hp/ou=rubycell cell.

Note

LDAP cell names can contain spaces or hyphens if they are enclosed in double quotes, but underscores are never allowed, even if they are enclosed in double quotes. For example, the LDAP cell names /c=us/o=hp /ou="excess cell" and /c=us/o=hp/ou="excess-cell" are allowed, but /c=us/o=hp/ou=excess_cell and /c=us/o=hp/ou="excess_cell" are not allowed.

Answer "Yes" to the question "Do you want to register the DCE cell in LDAP" during configuration of the cell. This puts the required DCE CDS information into the LDAP namespace for later use by GDA. No special LDAP client code is required on the host system—everything necessary was installed as part of the DCE installation. Refer to the documentation from your LDAP server vendor for more information about installing and configuring an LDAP server.

Execute an intercell command similar to the following command to show the root of the new cell and to see if everything works:

```
cdscp> show cell /.../c=us/o=hp/ou=rubycell
```

Enter the preceding command from an unauthenticated, nonprivileged account.

Now that you have configured and set up LDAP, you must use the Security Service rgy_edit cell command to create a cross-cell authentication account in the local and foreign cells. This account allows local principals to access objects in the foreign cell as authenticated users and vice versa.

In the /c=us/o=hp/ou=lasercell cell, you must use the rgy_edit cell command to create an account for /.../c=us/o=hp/ou=rubycell. (Refer to the Security Service commands in the *OSF DCE Administration Reference* for details about the cell command.) After adding the account for /.../c=us/o=hp/ou=rubycell in

the /c=us/o=hp/ou=lasercell cell, you should have an account entry that looks similar to the following:

```
krbtgt/c=us/o=hp/ou=rubycell [none none]:*:101:12::::
```

Note that the cell name is stripped of the path qualifier and is prefixed with krbtgt. The resulting name is used as the primary name for the cross-cell authentication account. You should now also have a principal entry that looks like the following:

```
krbtgt/c=us/o=hp/ou=rubycell     101
```

If a cell is reconfigured, changing its namespace and clearinghouse UUIDs, the krbtgt principal created by the cell command must be deleted using rgy_edit in the foreign cell. Note that for HP DCE for OpenVMS Alpha and OpenVMS I64 Version 3.2, the krbtgt principal must be deleted on both cells before the cell command is reexecuted between two cells. To test for proper configuration, show the cell information for the foreign cell. For example, in the /c=us/o=hp /ou=lasercell cell, use the cdscp show cell command to show information about the /c=us/o=hp/ou=rubycell cell. To do this at a laser cell host, execute the following command:

```
cdscp>  show cell /.../c=us/o=hp/ou=rubycell
```

To perform a similar operation from a /c=us/o=hp/ou=rubycell cell host, execute the following command:

```
cdscp> show cell /.../c=us/o=hp/ou=lasercell
```

9.4 Summary

The following steps summarize the intercell naming process. Refer to the chapter on managing intercell naming in the *OSF DCE Administration Guide — Core Components* for more information.

9.4.1 DNS Bind

For DNS bind:

1. Execute a cdscp show cell /.:/ as dns command.
2. Edit the hosts.db file and add the cell name.
3. Execute the kill -1 named-process-id command on UNIX systems to instruct the server to reread the database records. On an OpenVMS system acting as the DNS Master Bind Server, see the TCP/IP specific information for database update and nslookup instructions.
4. Perform an nslookup operation to verify that the cell information can be read. All three records (A, MX, and TXT) are returned.
5. Execute an intercell command similar to the following command to show the root of the new cell and to see if everything works:

   ```
   cdscp> show cell /.../ruby-cell.zko.dec.com
   ```

   Enter the preceding command from an unauthenticated, nonprivileged account.
6. Run dce_login, and log in as cell_admin.

7. Run rgy_edit, and execute a cell command similar to the following:

```
rgy_edit> cell /.../laser-cell.zko.dec.com

Enter group name of the local account for the foreign cell: none
Enter group name of the foreign account for the local cell: none
Enter org name of the local account for the foreign cell: none
Enter org name of the foreign account for the local cell: none
Enter your password:
Enter account id to log into foreign cell with: cell_admin
Enter password for foreign account:
Enter expiration date [yy/mm/dd or 'none']: (none) none
```

9.4.2 X.500

For X.500:

1. Answer "Yes" to the configuration question "Do you want to register the DCE cell in X.500".
2. Execute an intercell command similar to the following command to show the root of the new cell and to see if everything works:

   ```
   cdscp> show cell /.../c=us/o=hp/ou=rubycell
   ```

   Enter the preceding command from an unauthenticated, nonprivileged account.
3. Run dce_login, and log in as cell_admin.
4. Run rgy_edit, and execute a cell command similar to the following:

```
rgy_edit> cell /.../c=us/o=hp/ou=lasercell
Enter group name of the local account for the foreign cell: none
Enter group name of the foreign account for the local cell: none
Enter org name of the local account for the foreign cell: none
Enter org name of the foreign account for the local cell: none
Enter your password:
Enter account id to log into foreign cell with: cell_admin
Enter password for foreign account:
Enter expiration date [yy/mm/dd or 'none']: (none) none
```

9.4.3 LDAP

For LDAP:

1. Answer "Yes" to the configuration question "Do you want to register the DCE cell in LDAP".
2. Execute an intercell command similar to the following command to show the root of the new cell and to see if everything works:

   ```
   cdscp> show cell /.../c=us/o=hp/ou=rubycell
   ```

   Enter the preceding command from an unauthenticated, nonprivileged account.
3. Run dce_login, and log in as cell_admin.
4. Run rgy_edit, and execute a cell command similar to the following:

```
rgy_edit> cell /.../c=us/o=hp/ou=lasercell
Enter group name of the local account for the foreign cell: none
Enter group name of the foreign account for the local cell: none
Enter org name of the local account for the foreign cell: none
Enter org name of the foreign account for the local cell: none
Enter your password:
Enter account id to log into foreign cell with: cell_admin
Enter password for foreign account:
Enter expiration date [yy/mm/dd or 'none']: (none) none
```

10
Enhanced Browser

The Browser is a Motif-based tool for viewing the CDS namespace. The Browser can display an overall directory structure as well as show the contents of directories, enabling you to monitor growth in the size and number of directories in your namespace. You can customize the Browser so that it displays only a specific class of object names. The HP DCE Enhanced Browser contains some additional functions beyond those documented in the OSF DCE documentation.

10.1 Displaying the Namespace

When you start the Browser, an icon representing the root directory is the first item to appear in the window. Directories, soft links, and object entries all have distinct icons associated with them. Most object entries have unique icons based on their class; the class indicates the type of resource that the entry represents (for example, clearinghouse object entries). When the Browser does not recognize the class of an entry, it displays a generic icon. Figure 10–1 shows the Enhanced Browser icons and what they represent.

Figure 10–1 Enhanced Browser Icons

Icon	Entry Type
	Directory
	Object entry (generic)
	Soft Link
	Clearinghouse object entry
	Group

ZK−6001A−GE

10.2 Filtering the Namespace Display

Using the Filters menu, you can selectively display object entries of a particular class. With the Enhanced Browser, you can choose from either the RPC_Class or CDS_Clearinghouse object classes. For example, if you are interested in seeing the entries for clearinghouse objects only, choose the class CDS_Clearinghouse from the Filters menu. If you are interested in seeing object entries used in the

name service interface (NSI), choose RPC_Class. You can filter only one object class at a time.

Setting a filter does not affect the current display, but when you next expand a directory, you see only object entries whose class matches the filter. Note that soft links and directories still appear because only object entries can be filtered out. To reset the filter to view all object entries, choose the asterisk(*) from the Filters menu.

For a full description of the Browser, see the CDS section in the *OSF DCE Administration Guide — Core Components*.

11
IDL Compiler Enhancements

This chapter and the next two chapters describe enhancements to the IDL compiler supported by HP DCE for OpenVMS Alpha and OpenVMS I64.

This chapter describes the following enhancements:

- The `-standard` application build options
- Treatment of stub auxiliary files
- Use of HP Language-Sensitive Editor (LSE) Templates
- The binding handle callout feature

11.1 The -standard Build Option

The `-standard` IDL compiler command option allows you to specify portable or extended features of the OSF DCE.

The `standard_type` argument specifies what IDL features to enable. If you do not specify this argument, the compiler generates warning messages for all features that are not available in the previous version of OSF DCE.

You can specify one of the following values for the `standard_type` argument:

portable	Allows only the language features available in OSF DCE Version 1.0.2.
dce_v10, dce_v103, dec_v1.0	Synonymous with the portable argument.
dec_v13, dce_v11	Allows all language features supported by the `-standard dce_v10` argument, plus a set of HP extensions to its products based on OSF DCE Version 1.0.3.
extended	Allows all language features supported in the current version of the compiler. This is the default.
dce_v20	Synonymous with the extended argument.

The following example command line compiles the IDL interface `test.idl` and enables extended features of the OSF DCE:

```
% idl test.idl -standard extended
```

11.2 Stub Auxiliary Files

By default, the OpenVMS DCE IDL compiler does not generate stub auxiliary (AUX) files.

For applications that use certain data types or certain features of RPC, the current OSF DCE IDL compiler generates stub auxiliary files that contain support routines that are called by the stubs. The HP DCE implementation of the IDL compiler does not need those support routines, and, by default, does not generate stub auxiliary files.

However, if you are porting an RPC application from a platform on which the IDL compiler generates stub auxiliary files, and you do not want to modify your build procedures, you can set the IDL compiler to generate stub auxiliary files. To tell the IDL compiler to generate empty auxiliary files, define the symbol IDL_GEN_AUX_FILES with the following command:

```
$ IDL_GEN_AUX_FILES:== 1
```

11.3 HP Language-Sensitive Editor (LSE) Templates on OpenVMS

The IDL compiler supports the use of HP Language-Sensitive Editor (LSE) templates on OpenVMS to help you develop your interfaces more efficiently. LSE is a multilanguage, advanced text editor that enhances program development.

LSE provides the following features:

- Error Correction and Review

  This feature allows you to compile, review, and correct compilation errors within a single editing session. LSE provides an interface to the IDL compiler so that you can perform compilations without leaving LSE. The compiler provides LSE with compilation diagnostics in a way that allows you to review compilation errors in one editing window while displaying the related source in another window.

- Language-Specific Templates

  LSE accesses a collection of formatted language constructs, called templates, that provides keywords, punctuation, and placeholders for IDL.

While writing your program, you can use the COMPILE and REVIEW commands to compile your code and review compilation errors without leaving the editing session. The IDL compiler generates a file of compile-time diagnostic information that LSE uses to review compilation errors.

The LSE COMPILE command issues the appropriate command in a subprocess to invoke the IDL compiler and appends the /DIAGNOSTIC qualifier to the compile command.

The LSE REVIEW command displays any diagnostic messages that result from a compilation. LSE displays the compilation errors in one window, with the corresponding source code displayed in a second window. This multiwindow capability allows you to review your errors while examining the associated source code. This eliminates tedious steps in the error-correction process and helps ensure that you fix all errors before running the compilation process again.

See the LSE documentation for complete information on using LSE.

11.4 Binding Handle Callout

The binding handle callout feature lets you specify a routine that is automatically called from an IDL-generated client stub routine, in order to modify the binding handle.

You can typically use this feature to augment the binding handle with security context, for example, so that authenticated RPC calls are used between client and server.

This feature is particularly targeted at clients that use automatic binding via the auto_handle ACF attribute. For automatic binding, it is the client stub rather than the client application code that obtains a server binding handle. The binding handle q~callout feature lets you modify binding handles obtained by the client stub. Without this feature, you cannot modify the binding handles before the client stub attempts to initiate a remote procedure call to the selected server.

11.4.1 Attribute Configuration File

To select the binding handle callout feature, create an attribute configuration file (ACF) for the interface (if necessary) and place the binding_callout attribute on the interface. An example follows:

```
[auto_handle, binding_callout(my_bh_callout)] interface bindcall
{
}
```

The binding_callout attribute has the following general form:

```
[binding_callout(<IDENTIFIER> )]
```

You can specify the binding_callout attribute only once per interface; it applies to all operations in that interface.

11.4.2 Generated Header File

The IDL-generated header file for the interface contains a function prototype for the binding callout routine. In the previous example, bindcall.h contains a declaration similar to the following declaration:

```
void my_bh_callout(
    rpc_binding_handle_t *p_binding,
    rpc_if_handle_t interface_handle,
    error_status_t *p_st
);
```

11.4.3 Generated Client Stub

Each client stub routine in the IDL-generated client stub module calls the binding callout routine just before initiating the remote procedure call to the server. In the previous example, each client stub routine contains a call to my_bh_callout and passes the three arguments that are described in the following section.

11.4.4 Binding Callout Routine

The arguments to the binding callout routine are:

Input/Output:

rpc_binding_handle_t *p_binding

A pointer to a server binding handle for the remote procedure call. Generally, the binding callout routine augments this binding handle with additional context, such as for security.

Input:

rpc_if_handle_t interface_handle

The interface handle used to resolve a partial binding or for the binding callout routine to distinguish interfaces.

Output:

`error_status_t *p_st`

An error status code returned by the binding callout routine.

11.4.4.1 Error Handling

A binding callout routine returns `error_status_ok` when it successfully modifies the binding handle or decides that no action is necessary. This causes the client stub to initiate the remote procedure call.

When the binding callout routine returns an error status, the client stub does not initiate a remote procedure call. If `auto_handle` is being used, the client stub will attempt to locate another server of the interface and once again call the binding callout routine. Otherwise, the client stub executes its normal error handling logic.

A binding callout routine for a client using `auto_handle` can return rpc_s_no_more_bindings to prevent the client stub from trying to locate another server. The client stub will then execute its normal error handling logic.

By default, a client stub handles an error condition by raising an exception. If a binding callout routine returns an `rpc_s_` status code, the client stub raises the matching `rpc_x_` exception. If a binding callout routine returns any other error status, it is usually raised as an unknown status exception.

For an operation containing a `comm_status` parameter, the client stub handles an error condition by returning the error status value in the `comm_status` parameter. A binding callout routine can return any error status value to the client application code if the IDL operations are specified with `comm_status` parameters.

A binding callout routine can raise a user-defined exception rather than return a status code if it prefers to report application-specific error conditions back to the client application code via exceptions.

11.4.5 Predefined Binding Callout Routine

There is one predefined binding callout routine in the DCE library that may be suitable for some applications. To select this routine, specify a `binding_callout(rpc_ss_bind_authn_client)` ACF attribute.

`rpc_ss_bind_authn_client` matches the function prototype in the previous section, Generated Header File. It authenticates the client identity to the server, thereby allowing for one-way authentication. In other words, the client does not care which server principal receives the remote procedure call request, but the server verifies that the client is who the client claims to be.

`rpc_ss_bind_authn_client` does the following:

- Calls `rpc_ep_resolve_binding()` to resolve the binding handle if it is not fully bound (for example, for `auto_handle`).
- Calls `rpc_mgmt_inq_server_princ_name()` to obtain the server identity (principal name).
- Calls `rpc_binding_set_auth_info()` with all default values except the server principal name obtained in the previous call.
- If any of these calls returns an error status, places the error status in the `*p_st` argument and `rpc_ss_bind_authn_client` returns.

12

Application Debugging with the RPC Event Logger

The Remote Procedure Call (RPC) Interface Definition Language (IDL) compiler in HP DCE for OpenVMS Alpha and OpenVMS I64 includes enhanced application debugging support beyond the support provided with OSF DCE. The OpenVMS IDL compiler includes the RPC Event Logger, a software utility that records information about operations relating to the execution of an application. Operational information about the program state at a specific point during processing, called an event, is recorded in a file, called an event log. You have the option of directing event logging information to the terminal screen, rather than to a file. In this chapter, the terms event log and log are used interchangeably to refer to the stream of logging output captured in the event log file or displayed on the screen.

Event logging provides a detailed, low-level view of the execution of your RPC application. If development of your RPC application is proceeding well, this level of detail may not be necessary. However, when you are in the debugging phase of application development, the continuous execution information provided by the Event Logger and the ability to change the type and timing of logging can be valuable.

12.1 Introduction to the RPC Event Logging Facility

When event logging is enabled, the Event Logger creates one log for each client and server process. To enable the RPC Event Logger, you specify an IDL compiler option that traces events (described in Section 12.2.1).

Enabling event logging when compiling allows you the option of generating logs at runtime without rebuilding the application. Once logging is enabled, you can use OpenVMS symbols and the RPC Log Manager (`rpclm`) to control logging operations. The Log Manager provides a command interface for changing logging operations during application execution.

The RPC Event Logger records events about application calls, context handles, errors, miscellaneous events, and logging operations. These are called event types. Typical RPC events include the following:

- `call_start` — A client application made a call to a server.
- `call_failure` — A client stub terminated abnormally either through an exception or failing status.
- `exception` — An exception was detected in the server stub, and the exception caused the call to terminate.
- `context_rundown` — A context handle on a server was freed by the context rundown procedure.

For application calls, the Event Logger generates events that signal call activation, the call start and end, attempts to rebind to a server, and termination of a server thread.

For context handles, the Event Logger generates events that signal context handle creation and deletion by the client and server, and context handle modification, removal, and rundown.

For errors, the Event Logger generates events that signal call and receive failure from the client, exceptions, server failure, and call transmission failure from the server.

The miscellaneous events provide information about the application manager routine, and input and output argument processing events.

The logging operation itself generates events that display the logging output device, and that signal modification of logging parameters, and event log start and stop.

As a result of using the `-trace` option in the IDL compile command, `idl`, RPC events are generated by code in the client and server stub modules created by the compiler. Note that some events are generated at selected points in the RPC runtime library. For this reason, certain events, such as those relating to the logging operation, are always generated into the application code in addition to the event types you specify.

The events generated in each of these areas are shown in detail in Section 12.6.

In the event log, each event is described on a single line divided into five fields. The five fields are defined in Table 12–1.

Table 12–1 Event Log Fields

Field	Field Description
Event Time	The system clock at the time of the event. Events are listed chronologically in the log.
Thread Identity	The hostname, process ID, and thread ID.
Operation Name	The interface and operation name (if available).
Event Name	Name of the event.
Event Data	Data related to the event. This field contains either specific information about logging operations or a string binding that uniquely identifies the client process, server process, or Log Manager process.

The following is an example of an event log generated for an RPC client. The log contains five columns. To improve readability, columns four and five are shown below the first three columns. In addition, the field names have been added to identify the events; the names do not appear in an actual event log. (In subsequent event log examples, the field names are occasionally used instead of actual data to improve readability where necessary.)

```
EVENT TIME                              THREAD IDENTITY   OPERATION NAME

1999-02-07:11:48:18.31.160-5:00I0.121  ifdef:8710/1      binopwk.binopwk_add
1999-02-07:11:48:18.32.170-5:00I0.121  ifdef:8710/1      binopwk.binopwk_add
1999-02-07:11:48:18.65.180-5:00I0.121  ifdef:8710/1      binopwk.binopwk_add

EVENT NAME  EVENT DATA
```

```
log_start  all
call_start  ncacn_ip_tcp:16.31.48.109[1821]
call_end
```

This small event log indicates that the following events occurred:

1. The log_start event indicates that logging started on February 7, 1999, at 11:48 a.m. on the host named ifdef, in process number 8710, and in thread number 1. Event logging was enabled when the binopwk interface was compiled with the IDL -trace option. The RPC call to the binopwk_add operation in the binopwk interface caused logging to begin and is the first event logged. The Event Data field indicates that all events are being logged.
2. The call_start event indicates an attempt to execute a call to a server. The string binding in the Event Data field shows that the call was made over the TCP/IP transport to host 16.31.48.109 with endpoint 1821. This string binding identifies the server being contacted.
3. The call_end event indicates that the RPC call is completed, and control has returned to the caller of binopwk_add.

This log indicates that the RPC call to the binopwk_add interface was successful because no error events occurred.

12.2 Generating RPC Event Logs

In general, to create an event log, you must follow these four basic steps:

1. Specify the -trace option in your idl command line to enable event logging.
2. Compile and link the application.
3. Assign the event log to a filename or to the screen.
4. Run the application.

The next sections describe how to use the -trace option.

12.2.1 Enabling Event Logging

To enable event logging, you use a command line interface to the IDL compiler. The IDL compiler supports two interfaces:

- A universal interface that can be used on any operating system
- A Digital Command Language (DCL) interface that can be used only on the OpenVMS operating system

Your system manager determines which interface is available on your system. The following sections describe each interface. The examples use the universal interface to demonstrate event logging capabilities.

12.2.1.1 Universal IDL Compiler Interface

To enable event logging with the universal interface, specify the -trace option when you use the idl command to compile an interface. The syntax of the idl command with the -trace option is as follows:

```
$ idl filename -trace value
```

Event types are specified as a value of -trace. Valid values and the event types they denote are listed in Table 12–2.

Table 12–2 Event Values and Types

Value	Event Type
all	Log all events.
none	Disable all previously specified trace options.
calls	Log events relating to start and end of all RPC calls.
context	Log events relating to context handle creation, deletion, and rundown.
errors	Log errors.
misc	Log all miscellaneous events.
log_manager	Enable command interface support which allows modification at runtime of event logging options.

For more information about the `-trace` option, see Section 12.2.2.

12.2.1.2 Digital Command Language Interface for the Event Logger

This section defines the Digital Command Language (DCL) for the Event Logger.

NAME

IDL /TRACE — Invokes the Interface Definition Language (IDL) Compiler with event logging enabled.

SYNOPSIS

IDL *filename* /TRACE

QUALIFIER

/TRACE=*option*[,...])

Controls whether event logging is enabled. If you do not specify this qualifier, the compiler does not enable event tracing. To disable event logging, specify /NOTRACE.

Specify one or more of the following options:

LOG_MANAGER	Controls whether the Log Manager command line interface is enabled. The command line interface to the Log Manager allows you to modify event logging options at runtime. If you do not specify the LOG_MANAGER option, the command line interface will not be enabled. To disable the Log Manager, specify NOLOG_MANAGER.
EVENTS=*value*,...	Specifies the values for which event logging will be enabled. Specify one or more of the values shown in the following table, except for the value `log_manager`. (This function is provided by the DCL LOG_MANAGER qualifier.) If you specify only one option, you can omit the parentheses.

Table 12–3 lists some commonly used event logging options in the universal interface with DCL equivalents.

Table 12–3 Universal Interface with DCL Equivalents

Universal Interface	DCL Command
-trace all	/TRACE=EVENTS=ALL
-trace log_manager	/TRACE=LOG_MANAGER
-trace all -trace log_manager	/TRACE=(LOG_MANAGER,EVENTS=ALL)
-trace errors -trace calls	/TRACE=EVENTS=(ERRORS,CALLS)

12.2.2 Using the -trace Option

Once you have used the Event Logger, you will find that it generates a large volume of information to be analyzed. Discard any unneeded log files, since the Event Logger will continue to record information in the files, enlarging them until the disk is full.

To help reduce the generation of unwanted information, you can use the `-trace` options to enable event logging on only a subset of events. That is, rather than specifying the `all` option, specify only `calls` or only `context_handles`. The subset you specify will depend on the part of your application you are debugging. Because the `-trace` option provides logging control on a per-compilation basis, the interface must be rebuilt to enable or disable logging of different event types. The `-trace` options offer the ability to select different event types for the various IDL interfaces that may make up a single application.

You can use the `-trace` option to request logging of a single type of event, such as `errors`, with a command similar to the following:

```
$ idl binopwk.idl -trace errors
```

You can also use the `-trace` option to request logging of multiple event types, such as `errors` and `calls` as shown below:

```
$ idl binopwk.idl -trace errors -trace calls
```

This command enables the Event Logger, specifying error and call event logging.

To enable event logging to trace the execution of RPC calls within a process, perform the following steps:

1. Enable event logging by specifying the `-trace` option in the `idl` command you use to compile each interface definition. This example specifies the `-trace all` option:

   ```
   $ idl binopwk.idl -trace all
   ```

2. Build and link the client and server portions of the application.

3. Use the symbol RPC_LOG_FILE to direct the log output for both the server and client processes. To store Event Logger output in a file, assign the symbol to a filename. To direct Event Logger output to the standard terminal output for the process (stdout), assign the symbol to double quotation marks (" "). This guide refers to standard terminal output as the screen.

   In the window from which the server portion of the application will be executed, direct logging for the server to a file with the following syntax:

   ```
   $ RPC_LOG_FILE == "server.log"
   ```

   Or, to direct logging for the server to the screen, use the following syntax:

   ```
   $ RPC_LOG_FILE == ""
   ```

4. In the window from which the client portion of the application will be executed, direct logging for the client to a file with the following syntax:

   ```
   $ RPC_LOG_FILE == "client.log"
   ```

   Or, to direct logging for the client to the screen, use the following syntax:

   ```
   $ RPC_LOG_FILE == ""
   ```

Now you can invoke the client and server processes. The event log will be recorded in the specified file or displayed on your screen when you execute the application.

12.2.3 Combining Event Logs

Although event logs are generated locally for each process, you can combine event log files to provide a broader view of application execution.

Note that this section does not give examples of each step in the application development process.

The syntax of a merge command is as follows:

```
$ merge server-filename.log, client-filename.
log client_and_server-filename.log
```

If two events have the same timestamp, you receive a warning message after the merge is completed.

The following example illustrates how to merge logs from two different systems:

1. The server process command sequence is as follows:

   ```
   $ idl fpe_server.idl -trace calls -trace errors
   $ RPC_LOG_FILE == "server.log"
   $ server
   ```

2. The client process command sequence is as follows:

   ```
   $ idl fpe_client.idl -trace calls -trace errors
   $ RPC_LOG_FILE == "client.log"
   $ client
   ```

   These command sequences result in two log files: server.log and client.log, shown below. (Note that, in the following example log files, the Event Data field is replaced by the word <DATA> to improve readability of the log.)

   This is file server.log:

   ```
   1999-03-03:20:37.170-5:00I0.121 murp:17924/15 fpe.setup log_start <DATA>
   1999-03-03:20:37.170-5:00I0.121 murp:17924/15 RPC LogMgrlistening <DATA>
   1999-03-03:20:37.180-5:00I0.121 murp:17924/15 fpe.setup activate  <DATA>
   1999-03-03:20:37.180-5:00I0.121 murp:17924/15 fpe.setup terminate <DATA>
   1999-03-03:20:37.200-5:00I0.121 murp:17924/15 fpe.float          <DATA>
   1999-03-03:20:37.200-5:00I0.121 murp:17924/15     transmit_fault <DATA>
   1999-03-03:20:37.200-5:00I0.121 murp:17924/15 fpe.float terminate <DATA>
   ```

   This is file client.log:

   ```
   1999-03-03:20:37.850-5:00I0.121 ifdef:28168/1 fpe.stup log_start  <DATA>
   1999-03-03:20:37.880-5:00I0.121 ifdef:28168/1 fpe.stup call_start <DATA>
   1999-03-03:20:37.190-5:00I0.121 ifdef:28168/1 fpe.stup call_end   <DATA>
   1999-03-03:20:37.190-5:00I0.121 ifdef:28168/1 fpe.flt  call_start <DATA>
   1999-03-03:20:37.210-5:00I0.121 ifdef:28168/1      receive_fault <DATA>
   1999-03-03:20:37.210-5:00I0.121 ifdef:28168/1      call_failure  <DATA>
   ```

3. Next, the two log files are merged with the `merge` command:

```
$ merge server.log,client.log client_and_server.log
```

The resulting file `client_and_server.log` is as follows:

```
1999-03-03:20:37.850-5:00I0.121 ifdef:28168/1 fpe.setup log_start <DATA>
1999-03-03:20:37.880-5:00I0.121 ifdef:28168/1 fpe.setup call_start<DATA>
1999-03-03:20:37.170-5:00I0.121 murp:17924/15 fpe.setup log_start <DATA>
1999-03-03:20:37.170-5:00I0.121 murp:17924/15 RPC LogMgrlistening <DATA>

1999-03-03:20:37.180-5:00I0.121 murp:17924/15 fpe.setup terminate <DATA>
1999-03-03:20:37.190-5:00I0.121 ifdef:28168/1 fpe.setup call_end  <DATA>

1999-03-03:20:37.190-5:00I0.121 ifdef:28168/1 fpe.float call_start<DATA>
1999-03-03:20:37.200-5:00I0.121 murp:17924/15 fpe.float activate  <DATA>
1999-03-03:20:37.200-5:00I0.121 murp:17924/15 fpe.float exception <DATA>
1999-03-03:20:37.200-5:00I0.121 murp:17924/15      transmit_fault <DATA>
1999-03-03:20:37.200-5:00I0.121 murp:17924/15 fpe.float terminate <DATA>
1999-03-03:20:37.210-5:00I0.121 ifdef:28168/1       receive_fault <DATA>
1999-03-03:20:37.210-5:00I0.121 ifdef:28168/1       call_failure  <DATA>
```

For the merged output to be accurate, the system clocks on all hosts on which event logs are generated must be closely synchronized. The Distributed Time Service (DTS) component of HP's DCE provides such a service. Once the clocks are synchronized, the ordering of events in a merged log file is valid only if the difference between timestamps made on different machines is greater than the inaccuracy field in those timestamps. (See the DTS documentation for more information about timestamps.)

In the preceding `client_and_server.log` file example, consider the event with the timestamp 1999-03-03:20:37:03.180-5:00I0.121 and the event that follows it (these two event lines are separated from the rest of the log by a blank line before and after). Note that the timestamps indicate that the `terminate` event precedes the `call_end` event.

However, you cannot determine this sequence of events by comparing timestamps because the inaccuracy value at the end of the timestamp is greater than the difference between the timestamps. That is, the difference in time between these events is only 10 milliseconds (the difference between 180 and 190 milliseconds). However, the inaccuracy in the timestamps is 121 milliseconds (I0.121). Therefore, the log is not a definitive indicator of which event occurred first. Because of the simplicity of the example and the single thread of control, you can assume that the `terminate` event preceded the `call_end` event.

12.2.4 Disabling Event Logging

To disable event logging, compile your interface without specifying the `-trace` option. For example:

```
$ idl binopwk.idl
```

12.3 Using Symbols and the Log Manager to Control Logging Information

In addition to the `-trace` options, the Event Logger offers two other methods for controlling information in the event log. Each facility is advantageous in different circumstances, depending on the type of processes with which you are working and the type of events you need to log. The two methods are as follows:

- Controlling Logged Events with a Symbol: Select a subset of event types specified previously with the `-trace` option by creating the symbol RPC_

EVENTS. You assign the symbol to the required event types before executing the process. This method allows you to use event logging without rebuilding the interface; however, you must first stop the process or assign the symbol before starting it. This method is also useful in cases where you specified all-inclusive event logging (such as with the `-trace all` option) but you determine that you need only a subset of events while the application is executing.

- Controlling Logged Events with the RPC Log Manager: Select a subset of event types specified previously with the `-trace` option by using the RPC Log Manager command interface. This method allows you to modify event logging parameters for an executing image; there is no need to rebuild the interface or to stop and restart the process. In addition, you can use the Log Manager to modify event types specified with the symbol RPC_EVENTS.

The following sections provide detailed descriptions of how to use each of these methods to control the type of events logged.

12.3.1 Controlling Logged Events with a Symbol

One way to control the type of events logged is by assigning the symbol RPC_EVENTS. This method is ideal for an application that contains a single RPC interface because a symbol provides control at the process level, rather than at the interface-by-interface level. However, to enable the symbol, you must first stop the client or server process.

To use symbols to control event logging, first use the IDL `-trace` option in your `idl` compile command and then assign the log file with RPC_LOG_FILE. You can then use the symbol RPC_EVENTS to reduce the number of events currently being logged. That is, if you used the `-trace errors` option to request error event logging, you can subsequently use only the symbol to request logging of `errors` or `none`. You cannot use the symbol to increase the number of event types to be logged. To do this, you must recompile the interface with the required `-trace` options.

The value of RPC_EVENTS is a list of event types separated by commas. The list identifies the event types to be monitored. Valid values are the same as those for `-trace` (except `log_manager`). These values are `all`, `none`, `calls`, `context`, `errors`, and `misc`.

An example command line follows:

```
$ RPC_EVENTS == "calls,errors"
```

If the symbol RPC_EVENTS was not assigned, then by default all of the events specified with the `-trace` option are written into the event log.

12.3.2 Controlling Logged Events with the RPC Log Manager

During application development, certain problems occur only after a server has executed some number of calls. Other problems may require more information than anticipated to debug. These problems can be addressed by enabling the RPC Log Manager in your application image. The Log Manager offers a command line interface (`rpclm`) for manipulating logging operations during application execution. When you use the `rpclm` command line interface, you need not rebuild your interface or stop and restart your server or client process to manipulate logging operations.

The rpclm command interface commands are shown in Table 12–4.

Table 12–4 Command Interface to rpclm

Command	Description
inquire	Inquire about the currently logged events and determine the name of the active log file.
log	Specify additional events to log. Valid values are all, none, calls, context, errors, and misc.
unlog	Disable logging of the specified event types. Valid values are all, none, calls, context, errors, and misc.
file	Change the output device or file to which events are logged.
quit	Terminate the rpclm session.
help	Display a description of rpclm commands.

Follow these steps to enable the RPC Log Manager to control event logging:

1. Use the -trace log_manager option in your idl compile command.
2. Create the RPC_LOG_FILE symbol and assign it to a filename or to screen output.
3. Execute the client or server process, or both.
4. When the first call is made to an interface compiled with the -trace option, a listening event will be generated into the event log. Invoke the rpclm command interface (as described in step 4 which follows) by specifying the string binding from the listening event.

Note

Only string bindings from a listening event can be used to invoke rpclm.

The rpclm command interface allows you to control event logging parameters from your keyboard. You can use the command interface to reduce the events currently being logged as well as to manipulate logging operations. You can enable or disable logging of different event types (within the set selected with the -trace option), store event logging in a file or display it on the screen, inquire about the current event types being logged, and display the name of the current log file.

The following procedure illustrates how to use the Log Manager:

1. When you compile your interface with the idl compile option, include the -trace log_manager option. For example:

```
$ idl binopwk.idl -trace all -trace log_manager
```

2. Assign the RPC_LOG_FILE symbol to a filename. For example:

```
$ RPC_LOG_FILE == "client.log"
```

3. Execute the client or server process, or both.
4. Upon the first remote procedure call to an interface compiled with the -trace log_manager option, a listening event will be generated into the log. Examine the Event Data field of the listening event in the log to determine the Log Manager string binding. (The RPC Event Logger is itself a client

/server application: the Log Manager is a server process, and rpclm is its client. The rpclm client uses the string binding of the listening event to communicate with the Log Manager server.) Invoke rpclm and specify the Log Manager string binding. For example, consider the following event:

```
<TIME>  murp:17868/15  RPC LogMgr listening ncacn_ip_tcp:16.31.48.144[3820]
```

The listening event indicates that the RPC Log Manager is waiting for commands from rpclm. (Note that, in the example, the Time field is replaced by the word <TIME> to improve readability of the log.) To invoke rpclm, enter the listening event string binding for this server process from the Event Data field as follows:

```
$ rpclm "ncacn_ip_tcp:16.31.48.144[3820]"
```

Note that you must enclose the string binding in double quotation marks (" ").

5. As you execute rpclm commands, the Log Manager displays current logging parameters that indicate the changes made to event logging for this process. For example:

```
rpclm> unlog all
Event types:
Events logged to terminal
rpclm> log calls
Event types: call

Events logged to terminal
```

The log for this server process will have corresponding events logged as follows:

```
<TIME>   murp:17868/15   RPC Log Mgr         log_events           none
<TIME>   murp:17868/15   RPC Log Mgr         log_events           calls
```

The following example illustrates a command dialog between the user and rpclm. The dialog begins when the user specifies a string binding from a listening event to rpclm.

```
$ rpclm "ncacn_ip_tcp:cltdce[1821]"
rpclm> help
rpclm Commands:
inquire - Display logged events and log filename
log     - Specify additional events to log
unlog   - Specify events that should no longer be logged
file    - Change file into which events are logged
quit    - Exit log manager
rpclm> inquire
Event Types: calls
Events logged to terminal
rpclm> log errors
Event Types: calls errors
Events logged to terminal
rpclm> file server.log
Event Types: calls errors
Events logged to file 'server.log'
rpclm> quit
```

In this dialog, the user enters the help command to display the rpclm commands and command descriptions.

The user then enters the inquire command to display the types of events being logged and the log filename. In this example, errors are being logged to the screen.

The user enters the log calls command to specify that the Log Manager should start logging all events relating to calls, in addition to error events.

The user then enters the file command and specifies a filename. This command requests that rpclm change its output device from the terminal screen to a file named server.log.

The user then enters the quit command to terminate the rpclm session.

12.4 Using the -trace Option, Symbols, and the Log Manager Together

This section describes a few different ways to use the -trace options, symbols, and the Log Manager together. When you are learning to use the Event Logger, one possible approach is to specify all-inclusive event logging with the -trace all IDL compilation option, and then examine the event log to get an understanding of typical output. You can then use the symbol RPC_EVENTS to log only those events needed, such as calls or errors.

In the case of a running process that you do not want to terminate, use a different method. First enable the Event Logger specifying logging of all events, and enable the Log Manager also, as follows:

```
$ idl filename -trace all -trace log_manager
```

Set the event log to display on the screen, as follows:

```
$ RPC_LOG_FILE == ""
```

Then, assign the RPC_EVENTS symbol so it will not log any event types, as follows:

```
$ RPC_EVENTS == "none"
```

With these parameters set, the only event that will be displayed is the listening event once the first call is made to a server interface compiled with the -trace log_manager option. You can then obtain the string binding for the process and use it later, if needed. Once you start the process, if an error occurs, use the string binding to invoke the rpclm command interface and log the needed events. Any rpclm commands issued at this point will modify the RPC_EVENTS symbol assignment. For example, if you assign the symbol RPC_EVENTS to calls and then issue a command to rpclm to log errors, errors as well as calls will be logged.

Once you are familiar with Event Logger output, consider regularly using the command interface to enable or disable subsets of event types as needed.

This section provides an example of common tasks you may need to perform during event logging. In this particular example, a distributed server process provides a mathematical calculation service. The client process passes data to be calculated to the server process. This type of processing often generates exception events such as those in the example event log. That is, some operations are interrupted by floating point overflow and integer division by zero exceptions, as well as others. This example uses rpclm to control logging of a server process; however, rpclm can also be used to control event logging for a client process.

The following processes are shown in three windows: a server process window, a client process window, and an rpclm window.

1. Server Window — The user first enables the RPC Event Logger by specifying the -trace all and -trace log_manager options in the idl command line:

```
$ idl server_calc -trace all -trace log_manager
```

2. Server Window — The user starts the server process. The server receives a client call and initializes the RPC Log Manager. The symbols were assigned to enable event logging with no event types selected, so only Log Manager events are output, as shown. (Note that the end point displayed for the listening event is the end point of the Log Manager.)

```
$ RPC_LOG_FILE  == ""
$ RPC_EVENTS == "none"
$ server ncacn_ip_tcp

<TIME>  murp:17868/15 fpe.setup  log_start  none
<TIME>  murp:17868/15 RPC LogMgr listening  ncacn_ip_tcp:16.31.48.144[3820]
```

3. Client Window — The user invokes the client process by using a foreign command that was previously defined. The specified string binding is used to find the server. The client process displays the output PASS 1 upon completion.

```
$ client ncacn_ip_tcp 16.31.48.144 3820
PASS 1
```

4. rpclm Window — The user invokes rpclm and specifies the string binding displayed in the listening event output by the server process, shown in step 2. The string binding must be enclosed in double quotation marks (" "). The user issues the inquire command, and the event logging parameters for the server process are displayed. The Log Manager reply indicates that no event types are enabled and that the event log is being displayed on the screen from which the server process was started. The user issues the log errors command to enable logging of error events for the server process.

```
$ rpclm "ncacn_ip_tcp:16.31.48.144[3820]"
rpclm> inquire
Event types:
Events logged to terminal
rpclm> log errors
Event types: errors
Events logged to terminal
```

5. Client Window — The user invokes the client process a second time. The error events that occur during server execution are logged to the server window. The client process displays the output PASS 2 upon completion.

```
$ client ncacn_ip_tcp 16.31.48.144 3820
PASS 2
```

6. Server Window — The server process receives the command from rpclm to start logging errors. Any errors that occur in the server process are logged.

```
<TIME>  murp:17868/15 RPC Log Mgr log_events      errors
<TIME>  murp:17868/15 fpe.flt_overflw  exception  Floating point
                                                  overflow (dce/thd)
<TIME>  murp:17868/15             transmit_fault  rpc_s_fault_fp_overflow
<TIME>  murp:17868/15 fpe.flt_underflw exception  Floating point
                                                  underflow (dce/thd)
<TIME>  murp:17868/15             transmit_fault  rpc_s_fault_fp_underflow
<TIME>  murp:17868/15 fpe.flt_divbyzer exception  Floating point/decimal
                                                  divide by zero (dce/thd)
<TIME>  murp:17868/15             transmit_fault  rpc_s_fault_fp_div_by_zero
<TIME>  murp:17868/15 fpe.dble_overflw exception  Floating point
                                                  overflow (dce/thd)
<TIME>  murp:17868/15             transmit_fault  rpc_s_fault_fp_overflow
<TIME>  murp:17868/15 fpe.dble_underflw exception Floating point
                                                  underflow (dce/thd)
<TIME>  murp:17868/15             transmit_fault  rpc_s_fault_fp_underflow
<TIME>  murp:17868/15 fpe.dble_divbyzer exception Floating point/decimal
                                                  divide by zero (dce/thd)
<TIME>  murp:17868/15             transmit_fault  rpc_s_fault_fp_div_by_zero
```

7. rpclm Window — The user issues the unlog all command to disable logging of all previously specified event types.

```
rpclm> unlog all
Event types:
Events logged to terminal
```

8. Server Window — The event log now contains an entry that indicates the Event Logger will stop logging previously specified events.

```
<TIME>   murp:17868/15  RPC Log Mgr   log_events      none
```

9. rpclm Window — The user issues a log calls command to enable logging of call events.

```
rpclm> log calls
Event types: calls
Events logged to terminal
```

10. Server Window — The newest event log entry indicates that the Event Logger will start logging call events.

```
<TIME>    murp:17868/15 RPC Log Mgr   log_events   calls
```

11. rpclm Window — Because logging output will increase now that call events are being logged, the user issues an rpclm command to redirect logging output to a file named server_calc.log. When the application terminates and logging is complete, the user can use a text editor to view and search for entries in the log. This log file will contain only those call events from the server process.

```
rpclm> file server_calc.log
Event types: calls
Events logged to file 'server_calc.log'
```

12. Server Window — The newest event log entry indicates that the logger will start redirecting logging information to file server_calc.log.

```
<TIME>   murp:17868/15 RPC Log Mgr  log_file   server_calc.log
```

13. Client Window — The user invokes the client process a third time. The call events that occur during server execution are logged to file server_calc.log. The client process displays the output PASS 3 upon completion.

```
$ client ncacn_ip_tcp 16.31.48.144 3820
PASS 3
```

14. Server Log — This is log file server_calc.log:

```
$ TYPE server_calc.log

<TIME>  murp:17868/15 RPC Log Mgr       log_start server_calc.log
<TIME>  murp:17868/15 fpe.setup         activate  ncacn_ip_tcp:16.31.48.144[2905]
<TIME>  murp:17868/15 fpe.setup         terminate ncacn_ip_tcp:16.31.48.144[2905]
<TIME>  murp:17868/15 fpe.flt_overflw   activate  ncacn_ip_tcp:16.31.48.144[2905]
<TIME>  murp:17868/15 fpe.flt_overflw   terminate
<TIME>  murp:17868/15 fpe.flt_underflw  activate  ncacn_ip_tcp:16.31.48.144[2905]
<TIME>  murp:17868/15 fpe.flt_underflw  terminate
<TIME>  murp:17868/15 fpe.flt_divbyzer  activate  ncacn_ip_tcp:16.31.48.144[2905]
<TIME>  murp:17868/15 fpe.flt_divbyzer  terminate
<TIME>  murp:17868/15 fpe.dble_overflw  activate  ncacn_ip_tcp:16.31.48.144[2905]
<TIME>  murp:17868/15 fpe.dble_overflw  terminate
<TIME>  murp:17868/15 fpe.dble_underflw activate  ncacn_ip_tcp:16.31.48.144[2905]
<TIME>  murp:17868/15 fpe.dble_underflw terminate
<TIME>  murp:17868/15 fpe.dble_divbyzer activate  ncacn_ip_tcp:16.31.48.144[2905]
<TIME>  murp:17868/15 fpe.dble_divbyzer terminate
```

15. rpclm Window — The user issues a file command to redirect event logging output from server_calc.log to the terminal screen. To do this, press the Return key without specifying a filename when the Log Manager prompts for one.

```
rpclm> file
New File Name: <RETURN>
Event types: calls
Events logged to terminal
rpclm>
```

16. Server Window — The final event in the server_calc.log file is a log_file event, which indicates that event logging output is being redirected, in this case to the terminal screen. Therefore, no filename is displayed to the right of the event name.

```
<TIME>     murp:17868/15    RPC Log Mgr    log_file
```

12.5 Using Event Logs to Debug Applications

The RPC Event Logger is designed to help you debug your distributed application and is an enhancement over the basic diagnostics in the RPC product. The diagnostics alone provide minimal information. For example, the sample program called **test2**, which is provided with the HP DCE software kit, generates the rpc_x_no_more_bindings exception when the client fails to contact the server. Without the aid of RPC event logging, this is the only diagnostic information available.

The following example shows the basic RPC diagnostic information that an application displays when an error occurs:

```
$ run test2
%CMA-F-EXCCOPLOS, exception raised; some information lost
-DCERPC-E-NOMOREBINDINGS, no more bindings (dce / rpc)
*** Unable to obtain server binding information
$
```

If you enable RPC event logging by defining the symbol RPC_LOG_FILE, then the details of client execution can be captured in a file. From the event log, you can determine which servers the client tried to contact and the reason each attempt failed.

In the following event log example, the Event Data field on the rebind events indicates that the interface is not registered in the end point map and that a communications failure occurred. This information indicates that the server either is not running or it failed to register properly with the end point mapper.

The final event, call_failure, indicates that the call was terminated with the no more bindings status. This event indicates that the client tried all available servers but failed to communicate with any of them. (Note that in the first column the word <TIME> represents the actual value for time.)

```
$ run test2
<TIME>  ko:11436/1 test2.test2_add log_start  all
<TIME>  ko:11436/1 test2.test2_add call_start ncacn_ip_tcp:16.20.16.27[]
<TIME>  ko:11436/1 test2.test2_add rebind     not registered in endpoint
                                             map(dce/rpc)
<TIME>  ko:11436/1 test2.test2_add call_start ncacn_dnet_nsp:4.262[]
<TIME>  ko:11436/1 test2.test2_add rebind     not registered in endpoint
                                             map(dce/rpc)
<TIME>  ko:11436/1 test2.test2_add call_start ncadg_ip_udp:16.20.16.27[]
<TIME>  ko:11436/1 test2.test2_add rebind     communic failure (dce/rpc)
<TIME>  ko:11436/1 call_failure               no more bindings (dce/rpc)
%CMA-F-EXCCOPLOS, exception raised; some information lost
-DCERPC-E-NOMOREBINDINGS, no more bindings (dce / rpc)
*** Unable to obtain server binding information
$
```

12.6 Event Names and Descriptions

Table 12–5 lists and describes RPC events.

Table 12–5 RPC Events

Event Name	Description
activate	A thread was assigned to process an RPC call on a server, and the server stub has started processing input arguments. The Event Data field of the event log contains the string binding of the client application making the call.
await_reply	The transmission of input arguments in a call from a client application to a server is completed. The event is generated by the client stub. The client application is waiting for output arguments from the server.
call_end	A call from a client application is complete and the client stub is returning to the caller.
call_failure	A client stub terminated abnormally because either an exception occurred or a failing status was returned. The Event Data field of the event log contains the error text associated with the exception or RPC status code.
call_start	A client application attempted to make a call to a server. The event is generated by the stub within the client application. The Event Data field of the event log displays the string binding of the server being contacted.

(continued on next page)

Table 12–5 (Cont.) RPC Events

Event Name	Description
client_ctx_created	A client application has allocated a context handle on a particular server. The Event Data field of the event log contains the following information about this event: • The address representing the context handle in the client address space (an opaque pointer) • The UUID that can be used to identify the corresponding context handle on the server • The string binding of the server on which the actual context resided
client_ctx_deleted	The client application representation of a context handle is being deleted to reflect the deletion of the context handle on the server. The Event Data field of the event log contains the following information about this event: • The address representing the context handle in the client address space (an opaque pointer) • The UUID that can be used to identify the corresponding context handle on the server • The string binding of the server on which the actual context resided
client_ctx_destroyed	A client application has destroyed the client representation of a context handle through the `rpc_ss_destroy_client_context()` routine. The Event Data field of the event log contains the following information about this event: • The address representing the context handle in the client address space (an opaque pointer) • The UUID that can be used to identify the corresponding context handle on the server • The string binding of the server on which the actual context resided
context_created	A new context handle was created on a server and returned from the application manager routine. The Event Data field of the event log contains both the application value of the context handle and the UUID assigned to represent this context handle.
context_deleted	A context handle on a server has been deleted by the application manager routine. The Event Data field of the event log contains both the application value of the context handle and the UUID assigned to represent this context handle.

(continued on next page)

Table 12–5 (Cont.) RPC Events

Event Name	Description
context_modified	A context handle on a server was returned from the application manager routine with a value that is different from its previous value. The Event Data field of the event log contains both the application value of the context handle and the UUID assigned to represent this context handle.
context_rundown	A context handle on a server was freed by the context rundown procedure. The Event Data field of the event log contains both the application value of the context handle and the UUID assigned to represent this context handle.
exception	An exception was detected in the server stub, and the exception caused the call to terminate. The Event Data field of the event log contains a text description of the exception.
internal_error	A failure occurred in the support routines that manage the Event Logger. Check the Event Data field of the event log for a description of the cause of the event. If the error does not seem to indicate a transient network problem or an environmental failure, report the failure in a Software Performance Report (SPR).
listening	The RPC Log Manager has started to listen for `rpclm` commands. The `listening` event is generated by the portion of the RPC Log Manager built into your application by the RPC runtime when you specify the `-trace log_manager` option on your IDL compilation. The RPC Log Manager services the requests generated by the `rpclm` command. You use one of the string bindings from a `listening` event to invoke the `rpclm` command interface.
log_events	Event logging was modified through the Log Manager command interface `rpclm`. The Event Data field of the event log contains the new set of events being logged.
log_file	Event logging was modified through the Log Manager command interface `rpclm`. The Event Data field of the event log contains the new filename for the event log. If no filename is displayed, events are being logged to the screen.
log_start	A new event log was created or event logging was resumed after being suspended by a user command to the Log Manager command interface `rpclm`. The Event Data field in the event log contains a list of event types being logged.
log_stop	Event logging was stopped through the Log Manager command interface `rpclm`.
manager_call	The server stub is about to call the application manager routine.
manager_return	Control has just returned from the application manager routine to the server stub.

(continued on next page)

Table 12–5 (Cont.) RPC Events

Event Name	Description
rebind	A call from a client application to a server failed. The Event Data field in the event log shows the reason for the failure to contact the server. The event is generated by the stub within the client application. The call failed on an auto_handle operation and the client is attempting to rebind to the next server.
receive	Following the transmission of input arguments from a client application call to a server, the client received a reply and has started processing output arguments.
receive_fault	The client received a fault indicating a failure on the server. The Event Data field of the event log contains the RPC status that identifies the failure. All failures have fault codes that you can find in the file `ncastat.idl`. If the fault code in the `ncastat.idl` file is too general (such as `unspecified fault`), examine the server event log for precise failure information.
status_fail	A failure status was encountered in the server stub. The Event Data field of the event log describes the failure.
terminate	The server thread has completed processing the call and has terminated.
transmit_fault	The server runtime is sending fault information to the client application. The Event Data field of the event log indicates the name of the fault being sent. The fault information in this field is listed in the `ncastat.idl` file. The fault information in this field may be less descriptive than the information logged about the actual error. (See the `exception` or `status_fail` events in the event log to obtain precise failure information.)

13

Development of Distributed Applications with FORTRAN

This chapter explains how to use FORTRAN in the development of distributed applications that make remote procedure calls.

This chapter provides the following information:

- Interoperability and portability issues as they relate to applications written in FORTRAN
- A comprehensive example that introduces and illustrates several concepts
- General reference information about FORTRAN and remote procedure calls, including a discussion about restrictions

13.1 Interoperability and Portability

In general, an application you create in the HP DCE RPC environment will interoperate with other DCE RPC applications and will port to other DCE platforms if it complies with the appropriate programming language standards. More specifically:

- Any client that you have correctly created in a HP DCE RPC environment to use a DCE interface expressed in an IDL file will interoperate with any DCE RPC server that supports the interface.
- Any server that you have correctly created in a HP DCE RPC environment to use a DCE interface expressed in an IDL file will interoperate with any DCE RPC client that makes calls on the interface.

Typically, applications created in the DCE RPC environment are written in the C programming language. However, if you use the FORTRAN support in the HP DCE for OpenVMS Alpha and OpenVMS I64 software, the application will be subject to the following portability constraint:

- HP DCE RPC applications that contain code written in FORTRAN in an OpenVMS environment and that use a DCE interface expressed in an IDL file will interoperate with any corresponding DCE server or DCE client. However, you can port these applications only to other HP DCE environments.

13.2 Remote Procedure Calls Using FORTRAN — Example

The OpenVMS DCE IDL compiler provides similar support for applications written in FORTRAN as that provided for applications written in C. That is, you can write an RPC client in FORTRAN or you can write one or more manager routines in the server side of the application in FORTRAN. If you are unfamiliar with the tasks involved in developing an RPC application, see the chapter about application building in the *OSF DCE Application Development Guide*.

The FORTRAN support consists of stubs that use FORTRAN linkage conventions and a file that contains FORTRAN definitions of the constants and types declared in an interface definition. (These conventions and definitions are explained in Section 13.3)

The following sections present a comprehensive example that demonstrates how you can create the various parts of a simple, distributed payroll application using FORTRAN. The important features of this example are as follows:

- The example client application reads time-card information, passes it to a server that calculates wages, and prints the results.
- Both the client and the portion of the server that calculates gross pay (the manager routine) are written in FORTRAN.
- The initialization portion of the server application is written in C.

13.2.1 Where to Obtain the Example Application Files

All of the example application files referenced in this chapter are located in the following directory in your kit:

SYS$COMMON:[SYSHLP.EXAMPLES.DCE.RPC.PAYROLL]

Table 13–1 lists application files that normally would be created by the programmer for an application. To demonstrate application building, these application files are provided for you in the software kit. Table 13–2 in Section 13.2.3 lists the files generated by the IDL compiler for the example application.

Before you execute any of the example compilations, builds, or run commands in this chapter, copy all of the files listed in Table 13–1 to an empty directory. HP recommends that you read the file named PAYROLL.README in the same subdirectory. Then build and run the examples.

Table 13–1 Example Files Created by the Programmer

Filename	File Description
PAYROLL.IDL	The interface definition file that contains the application programming interface (API) to the remote procedure call `calculate_pay()`.
PRINT_PAY.FOR	The FORTRAN source file for the client side of the application.
SERVER.C	The FORTRAN source file that contains the initialization code for the server side of the application.
MANAGER.FOR	The FORTRAN source file for the server side of the application.
PAYROLL.COM	The command file that builds and runs the example application.
PAYROLL.DAT	The data input file for the example application.

The programs, procedures, and data files in the payroll example should be the same in this chapter and in the specified subdirectory that came with your HP DCE for OpenVMS Alpha or OpenVMS I64 software. For example, file PAYROLL.IDL as it appears in Section 13.2.2 should be identical to the following file:

SYS$COMMON:[SYSHLP.EXAMPLES.DCE.RPC.PAYROLL]PAYROLL.IDL

For all of the example files, if there is a difference between the file as shown in this chapter and the file in the subdirectory, assume that the file in the subdirectory is the correct one.

13.2.2 The Interface File and Data File (PAYROLL.IDL and PAYROLL.DAT)

The following interface, named PAYROLL.IDL, is part of the example application. The name of the remote procedure in the interface is `calculate_pay()`. The interface does not indicate that this procedure is written in FORTRAN.

```
/*
**               Copyright (c) 2004  by
**      Hewlett-Packard Development Company, Palo Alto, California.
**
*/

[
uuid(d1b14181-6543-11ca-ba11-08002b17908e),
version(1.0)
]
interface payroll
{
    const long string_data_len = 7;

    typedef struct {
        [string] char grade[string_data_len + 1];
    /* Storage for a string must include space for a null terminator */
        short   regular_hours;
        short   overtime_hours;
    } timecard;

    void calculate_pay(
        [in] timecard cards[1..7],
        [out] long *pay
    );
}
```

The next part of the example is the data file PAYROLL.DAT, which the client side of the application reads. The facts about each employee appear in eight records. The first record contains the employee's name (40 characters) and grade (7 characters). Records two to eight contain the number of regular hours and overtime hours worked on Monday to Sunday. Note that the time card structure defined in PAYROLL.IDL does not specify the employee's name in the data going to the remote procedure.

```
Jerry Harrison                          FOREMAN
 8 1
 8 1
 8 2
 8 2
 8 1
 0 4
 0 0
Tony Hardiman                           WORKER
 8 0
 8 0
 8 0
 8 2
 8 0
 0 4
 0 0
Mary Flynn                              WORKER
 8 1
 8 1
 8 2
 8 0
 8 1
 0 4
 0 0
```

13.2.3 Compiling the Interface with the IDL Compiler

To compile an RPC interface, you must invoke the IDL compiler. To compile an RPC interface for a FORTRAN application, you must select the following IDL options:

- Option `-lang fortran` (universal syntax) or the qualifier /LANGUAGE=FORTRAN (OpenVMS DCL syntax). This option specifies FORTRAN as the source code language.
- Option `-standard extended` (universal syntax) or the qualifier /STANDARD=EXTENDED (OpenVMS DCL syntax). This option enables features beyond those available in OSF DCE Version 1.0.3.

The following example commands illustrate how to invoke the IDL compiler using the universal and DCL interfaces, respectively, to compile the sample FORTRAN application interface:

```
$ idl payroll.idl -lang fortran -standard extended
$ IDL/LANGUAGE=FORTRAN/STANDARD=EXTENDED
PAYROLL.IDL
```

As a result of this command, the IDL compiler generates the files listed in Table 13–2.

Table 13–2 Example Files Created by IDL

Filename	File Description
PAYROLL_CSTUB.OBJ	The stub file generated by the IDL compiler for the client side of the application.
PAYROLL_SSTUB.OBJ	The stub file generated by the IDL compiler for the server side of the application.

(continued on next page)

Table 13–2 (Cont.) Example Files Created by IDL

Filename	File Description
PAYROLL.FOR	An include file that emulates the C language header file (.H) and that documents the valid syntax for subroutine calls that are used in the FORTRAN source files. This file will be called out by PAYROLL.COM and linked with the other application files because it refers to constants and types defined in the interface definition.
PAYROLL.FOR_H	A file generated by the IDL compiler that is used to build the stub files.

File PAYROLL.FOR, as generated by the IDL compiler, is next.

```
C     Generated by IDL compiler version HP DCE Vn.n.n-n
C
C     The following statements must appear in application code
C     INCLUDE 'NBASE.FOR'

      INTEGER*4 STRING_DATA_LEN
      PARAMETER (STRING_DATA_LEN=7)

      STRUCTURE /TIMECARD/
          CHARACTER*8 GRADE
          INTEGER*2 REGULAR_HOURS
          INTEGER*2 OVERTIME_HOURS
      END STRUCTURE

C      SUBROUTINE CALCULATE_PAY(CARDS, PAY)
C      RECORD /TIMECARD/ CARDS(7)
C      INTEGER*4 PAY
```

As you read this chapter, it is important to remember that the interface defined in file PAYROLL.IDL appears as FORTRAN statements in file PAYROLL.FOR. As a specific instance, consider the overtime hours field. Its definition appears in PAYROLL.IDL as the statement `short overtime_hours`, and in PAYROLL.FOR as the statement `INTEGER*2 OVERTIME_HOURS`. The overtime hours data in file PAYROLL.DAT is read into a data item of this type.

13.2.4 The Client Application Code for the Interface (PRINT_PAY.FOR)

Suppose that the directory in which the interface was compiled also contains file PRINT_PAY.FOR. This is the source file for the client side of the distributed application. Its contents follow.

```
C This is the client side of a payroll application that
C    uses remote procedure calls.
C
        PROGRAM PRINT_PAY
        INCLUDE 'PAYROLL.FOR'    ! Created by the IDL compiler from
                                 !    file PAYROLL.IDL.

C COPYRIGHT (C) 2004 BY HEWLETT-PACKARD DEVELOPMENT COMPANY. PALO ALTO, CALIFORNIA
```

```
C       The structure of a time card is described in the included file.
        RECORD /TIMECARD/ CARDS(7)
        CHARACTER*40 NAME
        CHARACTER*8  GRADE
        INTEGER*4 PAY
        INTEGER*4 I
C
C Read eight records for the current employee.
   10   READ (4, 9000, END=100) NAME, GRADE  ! First record
 9000   FORMAT (A40, A8)
        DO 20 I = 1, 7   ! Second through eighth records
           READ (4,9010) CARDS(I).REGULAR_HOURS, CARDS(I).OVERTIME_HOURS
 9010      FORMAT (I2, I2)
           CARDS(I).GRADE = GRADE
   20   CONTINUE
C
C Call remote procedure CALCULATE_PAY to calculate the gross pay.
        CALL CALCULATE_PAY (CARDS, PAY)
C Display the current employee's name and gross pay.
        WRITE  (6, 9020) NAME,     PAY
 9020   FORMAT (1X,      A40,  1X, I4  )
        GO TO 10
C
  100   STOP
C
        END
```

To compile and link the client program PRINT_PAY.FOR, which at runtime makes remote procedure calls to a server that supports the payroll interface, use the following DCL commands. After you enter the LINK command, press <CTRL/Z>.

```
$ FORTRAN PRINT_PAY.FOR
$ LINK PRINT_PAY, PAYROLL_CSTUB, DCE:DCE/OPT
```

Instead of using these two commands directly to build the client part of the application, you can invoke procedure PAYROLL.COM to build the entire application. See Section 13.2.8 for information about building and running this example.

This program reads its data from FORTRAN logical unit 4. A DCL command in procedure PAYROLL.COM defines the logical unit.

13.2.5 The Server Initialization File (SERVER.C)

Because all programming interfaces to the RPC runtime are specified in C, you must write the code that sets up the server in C. In this example, the server setup code (also called the initialization code) is in file SERVER.C.

This file is shown next. Both SERVER.C and PAYROLL.COM (shown in this section and in Section 13.2.8, respectively) contain the literal FORTRAN_payroll_mynode. Do not substitute a node name for mynode. The code in SERVER.C always exports its bindings using the entry name ".:/FORTRAN_payroll_mynode".

```
/* This is program SERVER.C that sets up the server for the application
   code whose origin is FORTRAN subroutine CALCULATE.PAY.              */

/*
** Copyright (c) 2004 by
**     Hewlett-Packard Development Company, Palo Alto, California.
**
*/
```

```
#include <STDIO.H>
#include <FILE.H>
#include <DCE/DCE_ERROR.H>
#include "payroll.for_h"  /* The IDL compiler created this file from
                             file PAYROLL.IDL.                      */

static char error_buf[dce_c_error_string_len+1];
static char *error_text(st)
    error_status_t st;
{
    error_status_t rst;
    dce_error_inq_text(st, error_buf, &rst);
    return error_buf;
}

main()
{
    error_status_t st;
    rpc_binding_vector_p_t bvec;

    /* Register all supported protocol sequences with the runtime. */
    rpc_server_use_all_protseqs(
        rpc_c_protseq_max_calls_default,
        &st
    );
    if (st != error_status_ok)
    {
     fprintf(stderr, "Can't use protocol sequence - %s\\n", error_text(st));
        exit(1);
    }

    /* Register the server interface with the runtime.                */
    rpc_server_register_if(
        payroll_v1_0_s_ifspec,  /* From the IDL compiler;             */
                                /* "v1_0" comes from the statement    */
                                /* "version(1.0)" in file PAYROLL.IDL. */
        NULL,
        NULL,
        &st
    );
    if (st != error_status_ok)
    {
        printf("Can't register interface - %s\\n", error_text(st));
        exit(1);
    }

    /* Get the address of a vector of server binding handles.  The
       call to routine rpc_server_use_all_protseqs() directed the
       runtime to create the binding handles.                      */
    rpc_server_inq_bindings(&bvec, &st);
    if (st != error_status_ok)
    {
        printf("Can't inquire bindings - %s\\n", error_text(st));
        exit(1);
    }
```

```
    /* Place server address information into the local endpoint map. */
    rpc_ep_register(
        payroll_v1_0_s_ifspec,
        bvec,
        NULL,
        (idl_char*)"FORTRAN Payroll Test Server",
        &st
    );
    if (st != error_status_ok)
    {
        printf("Can't register ep - %s\\n", error_text(st));
    }

    /* Place server address information into the name service database. */
    rpc_ns_binding_export(
        rpc_c_ns_syntax_default,
        (idl_char*)".:/FORTRAN_payroll_mynode",
        payroll_v1_0_s_ifspec,
        bvec,
        NULL,
        &st
    );
    if (st != error_status_ok)
    {
        printf("Can't export to name service - %s\\n", error_text(st));
    }

    /* Tell the runtime to listen for remote procedure calls.
       Also, FORTRAN cannot support multiple threads of execution. */
    rpc_server_listen((int)1, &st);
    if (st != error_status_ok)
        fprintf(stderr, "Error listening: %s\\n", error_text(st));

}
```

13.2.6 The Server Application Code for the Interface (MANAGER.FOR)

The server application code, written in FORTRAN, is declared in file PAYROLL.IDL as calculate_pay(). The file MANAGER.FOR provides some additional application code for the server and it contains subroutine CALCULATE_PAY as follows:

```
        SUBROUTINE CALCULATE_PAY(CARDS, PAY)
        INCLUDE 'PAYROLL.FOR'    ! Created by the IDL compiler from
                                 !    file PAYROLL.IDL.

C
C COPYRIGHT (C) 2004 HEWLETT-PACKARD DEVELOPMENT COMPANY. PALO ALTO, CALIFORNIA.

C       The structure of a time card is described in included file PAYROLL.FOR.
        RECORD /TIMECARD/ CARDS(7)
        INTEGER*4 PAY
        INTEGER*4 I

        PAY = 0
        DO 10 I = 1, 7
C     The basic hourly wage is $6.00.
            PAY = PAY + 6 * CARDS(I).REGULAR_HOURS
C     The following comparison does not include last character of GRADE,
C        because it arrives as a null terminator.
            IF (CARDS(I).GRADE(1:STRING_DATA_LEN) .EQ. 'FOREMAN') THEN
C         The overtime hourly wage for a foreman is $12.00.
                PAY = PAY + 12 * CARDS(I).OVERTIME_HOURS
            ELSE
C         The overtime hourly wage for a worker is $9.00.
                PAY = PAY +  9 * CARDS(I).OVERTIME_HOURS
            END IF
   10   CONTINUE
```

```
        RETURN
        END
```

To create the file SERVER.EXE, which at runtime responds to remote procedure calls from a client that supports the payroll interface, use the following DCL commands. After you enter the LINK command, press <CTRL/Z>.

```
$ CC SERVER
$ FORTRAN MANAGER
$ LINK SERVER, MANAGER, PAYROLL_SSTUB, DCE:DCE/OPT
```

Instead of using these commands directly to build the server part of the application, you can invoke procedure PAYROLL.COM to build the entire application (see Section 13.2.8).

13.2.7 Client and Server Bindings

To make remote procedure calls, client applications must be bound to server applications. This is illustrated in the client program PRINT_PAY.FOR shown in Section 13.2.4. The source code in the client program uses the default [auto_handle] binding, which is enabled by the following source code:

```
C Call remote procedure CALCULATE_PAY to calculate the gross pay.
        CALL CALCULATE_PAY (CARDS, PAY)
```

When you invoke procedure PAYROLL.COM (shown in Section 13.2.8), it displays a message about assuming [auto_handle].

For more information about client and server bindings, see the chapter on basic DCE RPC runtime operations in the *OSF DCE Application Development Guide*.

13.2.8 Building and Running the Example (PAYROLL.COM)

You can build, run, or both build and run the payroll example by using command file PAYROLL.COM. Its contents follow.

```
$!
$! This is file PAYROLL.COM to build, run, or both build and run
$!     the distributed payroll application.
$!

$!                        COPYRIGHT (C) 2004 BY
$!        HEWLETT-PACKARD DEVELOPMENT COMPANY. PALO ALTO, CALIFORNIA.
$!                        ALL RIGHTS RESERVED.

$!
$! THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
$! ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE INCLUSION
$! OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE OR ANY OTHER COPIES
$! THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY OTHER
$! PERSON.  NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
$!
$! THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE AND

$! SHOULD NOT BE CONSTRUED AS A COMMITMENT BY HEWLETT-PACKARD COMPANY.
```

```
$!
$! HP ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
$! SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY HP.
$!
$!
$! @PAYROLL        is the default -- to build and run.
$! @PAYROLL BUILD  does only the build.
$! @PAYROLL RUN    does only a sample run.
$!
$ SAY := WRITE SYS$OUTPUT
$ IF P1 .eqs. "RUN" then goto DO_RUN
$!
$! Build the application.
$ SAY "Building..."
$
$! Enable the universal IDL command interface
$ idl := $sys$system:dce$idl.exe
$
$! Compile the interface definition
$! -keep all is used to keep the IDL output for training purposes
$ idl PAYROLL.IDL -keep all -trace all -trace log_manager -lang fortran -
                  -standard extended
$
$! Compile the client application files
$ FORTRAN PRINT_PAY
$
$! Link the client application
$ LINK PRINT_PAY,PAYROLL_CSTUB, DCE:DCE/OPT
$
$! Compile the server application files
$ CC SERVER
$ FORTRAN MANAGER
$
$! Link the server application
$ LINK SERVER,MANAGER,PAYROLL_SSTUB,DCE:DCE/OPT
$ IF P1 .eqs. "BUILD" then exit
$
$DO_RUN:
$! Run the application.
$ SAY "Activating server image..."
$ DEFINE/NOLOG RPC_DEFAULT_ENTRY ".:/FORTRAN_payroll_mynode"
$  SPAWN/NOWAIT/INPUT=NL:/OUTPUT=SERVER.LOG/PROCESS=FORTRAN_SERVER -
        RUN SERVER
$ WAIT 00:00:10     ! Allow 10 seconds for the server to start.
$ DEFINE/NOLOG FOR004 PAYROLL.DAT
$ SAY "Activating client image..."
$  RUN PRINT_PAY
$ SAY "Deleting server process..."
$ STOP FORTRAN_SERVER
$ SAY "End of sample application"
```

If you prefer to use OpenVMS DCL syntax for the command to the IDL compiler, then one way to modify file PAYROLL.COM follows.

Consider these seven lines:

```
$! Enable the universal IDL command interface
$ idl := $sys$system:dce$idl.exe
$
$! Compile the interface definition
$! -keep all is used to keep the IDL output for training purposes
$ idl PAYROLL.IDL -keep all -trace all -trace log_manager -lang fortran -
                  -standard extended
```

Change the second, third, sixth, and seventh lines to comments by adding a "$!" at the beginning of each line (or delete all seven lines). Then, add the following three lines before the two lines:

```
$
$! Compile the client application files

$! Use OpenVMS DCL syntax for the command to the IDL compiler.
$ IDL/KEEP=ALL/TRACE=(EVENTS=ALL,LOG_MANAGER)/LANGUAGE=FORTRAN -
     /STANDARD=EXTENDED PAYROLL.IDL
```

This modification works only if you do not otherwise have the symbol IDL defined.

13.2.9 Example Output

The output from building and running the sample application looks like this:

```
Building...
Operation calculate_pay has no binding handle parameter; [auto_handle] assumed
Activating server image...
%DCL-S-SPAWNED, process FORTRAN_SERVER spawned
Activating client image...
Jerry Harrison                          372
Tony Hardiman                           294
Mary Flynn                              321
FORTRAN STOP
Deleting server process...
End of sample application
```

The next time you need to run this application, enter this command:

```
$ @PAYROLL RUN
```

The output from building this application includes files PRINT_PAY.EXE (client) and SERVER.EXE (server). You can use these executable programs in separate client and server processes.

13.3 Remote Procedure Calls Using FORTRAN — Reference

Section 13.2 contains a comprehensive example that introduces creating distributed applications with FORTRAN program units. This section goes beyond the example to provide reference information and explain general concepts about creating these distributed applications.

13.3.1 The FORTRAN Compiler Option

If you are generating stubs and include files for application code written in FORTRAN, you must specify it as the language of choice when you compile the application's IDL file. To specify the FORTRAN language using the universal syntax, specify `-lang fortran`; the default value is `-lang c`. To specify the FORTRAN language using DCL syntax, specify /LANGUAGE=FORTRAN; the default value is /LANGUAGE=CC.

In the remainder of this chapter, the phrase "FORTRAN option" refers to the IDL command that specifies FORTRAN. Examples of the IDL command and specification are presented in Section 13.2.3.

Any client or server stub files that the FORTRAN option generates use the FORTRAN linkage conventions. This means that all parameters are passed by reference (see Section 13.3.5.1 for more information). In addition, all identifiers are converted to uppercase.

The FORTRAN option generates the file *filename*.FOR, which includes FORTRAN declarations of the constants and types declared in the IDL file. The .FOR file also includes, for each operation declared in the IDL file, a set of comments that describes the signature of the operation in FORTRAN terms.

In addition, the FORTRAN option generates the file *filename*.FOR_H. This file is used for generating the client and server stubs. It is also needed for generating FORTRAN stubs for any interface that imports this interface.

Consider the header option whose syntax is `-header` (universal) or /HEADER= (OpenVMS DCL). If you specify both the FORTRAN option and the header option to the IDL compiler, the following rules govern the compiler's placement of the files *filename*.FOR and *filename*.FOR_H.

- If you specify a directory name in the header option, the compiler places the files in that directory. Otherwise, it places the files in the current default directory.
- If you specify a filename without an extension in the header option, the compiler uses that filename with the extensions .FOR and .FOR_H.
- If you specify a filename with an extension in the header option, the compiler uses that file extension instead of .FOR_H; however, the compiler does not change the extension of the .FOR file.

13.3.2 Restrictions on the Use of FORTRAN

This section discusses restrictions on distributed applications written in FORTRAN that make remote procedure calls. These restrictions are on interfaces and stubs, and on runtime operations.

- If an interface contains any arrays that have more than seven dimensions, the IDL compiler cannot produce output that is compatible with FORTRAN.
- If an interface contains two identifiers that differ only in the case of their characters, the IDL compiler may not be able to build stubs.
- The stubs generated for FORTRAN cannot call operations that use pipes.
- If the `transmit_as` or `represent_as` attributes have been applied to a character array type used to define the parameters of an operation, then FORTRAN cannot call that operation.
- If the `transmit_as` or `represent_as` attributes have been applied to an array type that, in turn, is the base type of an array type used to define the parameters of an operation, then FORTRAN cannot call that operation.
- If the `v1_array` attribute has been applied to any parameter of an operation, then FORTRAN cannot call that operation.
- FORTRAN does not allow the concurrent execution of two or more threads. In particular, if a server implements remote operations in FORTRAN, it must restrict the number of threads of server execution to 1. The following statement in file SERVER.C (shown in Section 13.2.5) specifies this restriction:

```
rpc_server_listen((int)1, &st);
```

13.3.3 IDL Constant Declarations

A constant declaration either gives a name to an integer or string constant or gives a second name to a constant that has already been given a name. Examples of these declarations follow:

```
const long array_size = 100;
const char jsb = "Johann \\"Sebastian' Bach";
const long a_size = array_size;
const boolean untruth = FALSE;
```

For all IDL constant declarations, equivalent PARAMETER statements are generated in the corresponding file *filename*.FOR. For example:

```
INTEGER*4 ARRAY_SIZE
PARAMETER (ARRAY_SIZE=100)

CHARACTER*(*) JSB
PARAMETER (JSB='Johann "Sebastian'' Bach')

INTEGER*4 A_SIZE
PARAMETER (A_SIZE=ARRAY_SIZE)

LOGICAL*1 UNTRUTH
PARAMETER (UNTRUTH=.FALSE.)
```

All integer constants are declared as INTEGER*4.

All `void *` constants are ignored.

A nonprinting character that appears within a character or string constant is replaced by a question mark (?).

13.3.4 Type Mapping

An IDL type that is a synonym for another type is presented to FORTRAN as the type for which the synonym is defined. For example, suppose that the IDL file contains the following statement:

```
typedef foo bar;
```

Then, all instances of IDL type *Cbar) are presented to FORTRAN as of type `foo`.

Table 13–3 describes the mappings from IDL types to FORTRAN types:

Table 13–3 Mappings for IDL Types

IDL Data Type	FORTRAN Data Type	Comments
arrays		See notes 8 and 9
boolean	LOGICAL*1	
byte	BYTE	
char	CHARACTER	
context handle	INTEGER*4	
double	REAL*8	See note 3
enum	INTEGER*4	
error_status_t	INTEGER*4	See note 4

(continued on next page)

Table 13–3 (Cont.) Mappings for IDL Types

IDL Data Type	FORTRAN Data Type	Comments
float	REAL*4	
handle_t	HANDLE_T	See description of NBASE.FOR in this chapter
hyper	IDL_HYPER_INT	See description of NBASE.FOR
ISO_MULTI_LINGUAL	ISO_MULTI_LINGUAL	See description of NBASE.FOR
ISO_UCS	ISO_UCS	See description of NBASE.FOR
long	INTEGER*4	
pipe		No mapping
pointer	INTEGER*4	See note 10
short	INTEGER*2	
small	INTEGER*2	See note 1
struct	STRUCTURE	See notes 5 and 6
union	UNION	See note 7
unsigned hyper	IDL_UHYPER_INT	See description of NBASE.FOR
unsigned long	INTEGER*4	See note 2
unsigned short	INTEGER*4	See note 1
unsigned small	INTEGER*2	See note 1

Notes

1. For these IDL data types, the FORTRAN data type is chosen because it can represent all possible values of the IDL type. Note that, in each case, there are values of the FORTRAN type which cannot be represented in the IDL type. You must not attempt to pass such values in parameters. The RPC runtime code does not perform range checking.
2. Because some values that can be represented in an IDL data type cannot be represented correctly in the FORTRAN data type, the IDL compiler issues a warning.
3. You must compile FORTRAN code that uses this data type and specify the /G_FLOAT compiler option.
4. Status code mapping will occur where necessary.
5. For any structure type in the IDL file that is not defined through a typedef statement, the IDL compiler generates the name of the FORTRAN structure. To determine what name was generated, look at *filename* .FOR.
6. The semantics of conformant structures cannot be represented in FORTRAN. In the definition of such a structure in *filename*.FOR, a placeholder for the conformant array field is specified as a one-dimensional array with one element. If the first lower bound of the conformant array is fixed, this value is used as the lower and upper bounds of the placeholder. If the first lower bound of the array is not fixed and if the first upper bound of the conformant array is fixed, the upper bound is used as the lower and upper bounds of the

placeholder. Otherwise, the lower and upper bounds of the placeholder are zero.

7. Note that IDL encapsulated union types and nonencapsulated union types are represented as FORTRAN structures containing unions.
8. IDL array types are converted to arrays of a nonarray base type.
9. Arrays that do not have a specified lower bound have a lower bound of zero. Consider the following two statements in an IDL file:

```
double d[10][20];
short e[2..4][3..6];
```

The statements map into the following FORTRAN constructs:

```
REAL*8 D(0:9,0:19)
INTEGER*2 E(2:4,3:6)
```

10. The size of the pointer depends on the platform. It is INTEGER*4 for OpenVMS systems and INTEGER*8 for HP Tru64 UNIX Alpha systems.

13.3.5 Operations

Operations can pass parameters and return function results. This section explains these topics.

13.3.5.1 Parameter Passing by Reference

The following rules explain the mapping between IDL parameters and FORTRAN parameters:

- If the IDL parameter contains an asterisk (*) and does not have a [ptr] or [unique] attribute, this signifies a parameter of the indicated type passed by reference. The FORTRAN parameter is of the same type.
- If the IDL parameter contains an asterisk and does have a [ptr] or [unique] attribute, the FORTRAN parameter is a pointer.
- If the IDL parameter is an array and has the [ptr] or [unique] attribute, the FORTRAN parameter is a pointer.
- If none of the preceding cases is true, then the FORTRAN parameter is of the same type as the IDL parameter.

13.3.5.2 Function Results

The only possible function result types in FORTRAN are scalars and CHARACTER*n. The mappings from IDL to FORTRAN never produce CHARACTER*n, where n is greater than 1.

IDL hyper integers are not scalars in terms of function results, but IDL pointers are treated as scalars because they are mapped to INTEGER*4.

For an operation that has a result type that is not allowed by FORTRAN, the stubs treat the operation result as an extra [out] parameter added to the end of the parameter list.

If the type of an operation is not void, you must state the type of the function result in FORTRAN.

13.3.6 Include Files

Usually, a FORTRAN routine that is part of an RPC client or manager for the interface defined in *filename*.IDL must include the following files:

- *filename*.FOR
- NBASE.FOR
- The .FOR files for any imported interfaces

Program units PRINT_PAY.FOR and MANAGER.FOR (containing subroutine subprogram CALCULATE_PAY) in the example of a distributed payroll application do not include NBASE.FOR because the units contain none of the IDL data types in Table 13–2. Otherwise, the program units would include NBASE.FOR. Furthermore, these units could safely include NBASE.FOR even though it is unnecessary in the example.

13.3.7 The NBASE.FOR File

DCE:NBASE.FOR declares standard data types used in mapping IDL to FORTRAN. The declarations include those listed in Table 13–4.

Table 13–4 Standard Declarations

IDL Data Type	FORTRAN Declaration	Comments
hyper	STRUCTURE /IDL_HYPER_ INT/	
	INTEGER*4 LOW	
	INTEGER*4 HIGH	
	END STRUCTURE	
unsigned hyper	STRUCTURE /IDL_UHYPER_ INT/	
	INTEGER*4 LOW	
	INTEGER*4 HIGH	
	END STRUCTURE	
handle_t	STRUCTURE /HANDLE_T/ INTEGER*4 OPAQUE_ HANDLE END STRUCTURE	Size of pointer is platform specific: INTEGER*4 on OpenVMS systems INTEGER*8 on HP Tru64 UNIX Alpha systems
ISO_MULTI_ LINGUAL	STRUCTURE /ISO_MULTI_ LINGUAL/	

(continued on next page)

Table 13–4 (Cont.) Standard Declarations

IDL Data Type	FORTRAN Declaration	Comments
	BYTE ROW	
	BYTE COLUMN	
	END STRUCTURE	
ISO_UCS	STRUCTURE /ISO_UCS/	
	BYTE GROUP	
	BYTE PLANE	
	BYTE ROW	
	BYTE_ COLUMN	
	END STRUCTURE	

13.3.8 IDL Attributes

This section describes IDL attributes that apply to RPC applications containing FORTRAN modules.

13.3.8.1 The transmit_as Attribute

The presented type must be expressible in FORTRAN. Because addresses are involved, the routines used for data conversion cannot be written in VAX FORTRAN.

13.3.8.2 The string Attribute

A FORTRAN data item corresponding to an IDL string contains the number of characters specified for the IDL string. Because IDL strings are usually terminated with a null byte, the following transmission rules apply:

- If a FORTRAN routine contains data for transmission, and a null byte appears before the last character of the FORTRAN data item, then the characters up to and including the null byte are transmitted.
- If a FORTRAN routine contains data for transmission, and a null byte does not appear before the last character of the FORTRAN data item, then all the characters of the data item except the last are transmitted, followed by a null character.
- If data is transmitted to a FORTRAN routine, then the FORTRAN data item receives a null terminated string. If the FORTRAN data item contains more characters than the string, then the additional characters are not affected.

An IDL operation can have a conformant string parameter. Such a parameter is presented to FORTRAN as type CHARACTER*(*). If the base type of the string consists of *w* bytes and the string consists of *n* characters, then the parameter size is *n***w*. The maximum parameter size supported is 65535.

A conformant string field of a structure will have type CHARACTER**w*, where *w* is the number of bytes in the base type of the string.

In all other cases where a string is not the target of a pointer, the IDL file specifies the string. Such a string is presented to FORTRAN as CHARACTER*s, where *s* is the product of the string length and the number of bytes in the base type of the string. Furthermore, *s* must be between 1 and 65535 inclusive.

13.3.8.3 The context_handle Attribute

A context handle rundown routine cannot be written in FORTRAN because the routine must handle address information.

13.3.8.4 The Array Attributes on [ref] Pointer Parameters

A `[ref]` pointer parameter that has array attributes attached to it is presented to FORTRAN as the equivalent array.

13.3.9 ACF Attributes

The following items can occur in an Attribute Configuration File (ACF). They require special consideration when you are using FORTRAN.

13.3.9.1 The implicit_handle ACF Attribute

You must supply a COMMON block whose name is the name given in the implicit handle clause. This COMMON block must contain the binding handle as its only data item.

For example, suppose an ACF contains the following interface attribute:

```
[implicit_handle(handle_t i_h)]
```

Then, any FORTRAN routine that calls an operation which uses the implicit binding must include statements with the following form:

```
RECORD /HANDLE_T/ BINDING_HANDLE
COMMON /I_H/      BINDING_HANDLE
```

13.3.9.2 The represent_as ACF Attribute

The local type must be expressible in FORTRAN. Because addresses are involved, you cannot write the data conversion routines in FORTRAN.

A type name in a `represent_as` attribute that does not occur in the interface definition and is not an IDL base type is assumed to be a STRUCTURE type.

Suppose that the `represent_as` type is not an IDL base type or a type defined in your IDL source. Then, you must supply a `.h` file whose unextended name is given in an include statement in the ACF. (An unextended name is a filename without the file extension that follows the dot (.) in the name. For example, the unextended filename for file EXAMPLE.H is EXAMPLE.) This file must include a definition of the local type in C syntax. You will need a *filename*.FOR file containing a FORTRAN definition of the local type. HP recommends that you assign this file the same unextended name.

14
Troubleshooting

This chapter provides help for tracking down problems you may have with HP DCE for OpenVMS Alpha and OpenVMS I64.

14.1 General Troubleshooting Steps

If you are experiencing problems with DCE on your system, go through the following steps to help you isolate the problem:

1. Use the DCE$SETUP SHOW option to examine the current state of the DCE services on your system:

   ```
   $ @SYS$MANAGER:DCE$SETUP SHOW
   ```

   This command tells you how DCE is configured on your system, what daemons should be active, and what daemons are currently active.

2. Make sure that your system has the correct DCE configuration. If not, you may need to repeat the CONFIGURE operation of DCE$SETUP:

   ```
   $ @SYS$MANAGER:DCE$SETUP CONFIGURE
   ```

3. You should also use the DCL command:

   ```
   $ SHOW SYSTEM
   ```

   to ensure that the TCP/IP Internet ACP process (INET_ACP) is running. If it is not running, you may need to restart TCP/IP services on your system with the following commands:

   ```
   $ @SYS$STARTUP:TCPIP$SHUTDOWN
   $ @SYS$STARTUP:TCPIP$STARTUP
   ```

4. If the DCE$SETUP SHOW command indicates that a configured DCE daemon is not currently running on your system, check for component-specific output, error, and log files (*.OUT, *.ERR, *.LOG) in the following directories:

   - Security — DCE$SPECIFIC:[VAR.SECURITY]
   - CDS — DCE$SPECIFIC:[VAR.DIRECTORY.CDS]
   - NSID — DCE$SPECIFIC:[VAR.DIRECTORY]
   - DTS — DCE$SPECIFIC:[VAR.ADM.TIME]
   - RPC — DCE$SPECIFIC:[VAR.DCED]

5. It may be possible to start up missing DCE daemons with the START option:

   ```
   $ @SYS$MANAGER:DCE$SETUP START
   ```

6. If DCE daemons will not start properly, try a RESTART operation. This STOPs all daemons and STARTs them again in an orderly fashion:

   ```
   $ @SYS$MANAGER:DCE$SETUP RESTART
   ```

7. If problems persist, try a CLEAN followed by a START operation. This will delete the temporary DCE databases and restart the daemons in an orderly fashion:

```
$ @SYS$MANAGER:DCE$SETUP CLEAN !This will stop all daemons
$ @SYS$MANAGER:DCE$SETUP START
```

8. Ensure that all files and directories in the DCE$SPECIFIC: directory tree have the proper owner and protection:

```
[VAR.SECURITY]CREDS.DIR              [DCE$SERVER]   (RWE,RWE,RWED,RWED)
All other directories:               [DCE$SERVER]   (RWE,RWE,RE,RE)
[ETC.SECURITY]PE_SITE.;              [DCE$SERVER]   (RWED,RWED,RWED,RE)
[KRB5]V5SRVTAB.;                     [DCE$SERVER]   (RWD,RWD,,)
[VAR.ADM.DIRECTORY.CDS]CDS_CACHE.*   [DCE$SERVER]   (RWD,RWD,,)
[VAR.ADM.DIRECTORY.CDS]CLERK_MGMT_ACL.DAT
[DCE$SERVER]       (RWD,RWD,R,)
[VAR.ADM.TIME]MGT_ACL.DAT;1          [DCE$SERVER]   (RWD,RWD,R,)

[VAR.SECURITY]SEC_CLIENTD.BINDING;1  [DCE$SERVER]   (RWD,RWD,R,R)
[VAR.SECURITY.CREDS]*.NC             [DCE$SERVER]   (RWED,RWED,RWED,)
All other [VAR.SECURITY.CREDS] files [DCE$SERVER]   (RWD,RWD,,)
```

9. Ensure that all files and directories in the DCE$COMMON: directory tree have the proper protection:

```
 All [ETC] files and directories             (RWED,RWED,RWED,RE)
 All [ETC.DIRECTORY] files and directories   (RWED,RWED,RWED,RE)
 All [ETC.SECURITY] files and directories    (RWED,RWED,RWED,RE)
```

10. If DCE on the security registry server system for your cell is reconfigured, you must reconfigure all OpenVMS client systems in the cell.

14.2 Time Problems During Configuration

This section discusses problems with time and time zones that you may encounter during configuration or startup.

14.2.1 Time Zone Configuration

During DCE configuration or startup, you may encounter the following message:

```
Error: UTC services and run-time library don't agree on the local time
%SYSTEM-F-ABORT, abort
```

This message indicates that your current time zone configuration is invalid. Verify the definition of the logical names used by UTC services by entering the DCL command:

```
$ SHOW LOGICAL SYS$*TIME*
```

You should see five logical names listed:

- SYS$TIMEZONE_DAYLIGHT_SAVING should be 0 if this current date and time are not during daylight savings time, or 1 if it is.
- SYS$TIMEZONE_DIFFERENTIAL should be the difference in seconds between the local time and Greenwich Mean Time (GMT).
- SYS$TIMEZONE_NAME is the name of the current time zone.
- SYS$TIMEZONE_RULE is a complex string representing the possible time zones for the current location, including the time zone names and the time zone differentials during daylight savings time and during standard time.

- SYS$LOCALTIME should point to the file containing the time zone information for your local time zone (for example, SYS$SYSROOT:[SYS$ZONEINFO.SYSTEM.US]EASTERN).

If these logicals appear to be incorrect, reconfigure your time zone information as follows:

```
$ DEASSIGN /SYS /EXEC SYS$TIMEZONE_RULE
$ DELETE SYS$SYSTEM:DTSS$TIMEZONE_DIFFERENTIAL.DAT;*
$ DELETE SYS$STARTUP:DTSS$UTC_STARTUP.COM;*
$ @SYS$MANAGER:UTC$TIME_SETUP
$ @SYS$UPDATE:DTSS$INSTALL_TIMEZONE_RULE
```

The procedure DTSS$INSTALL_TIMEZONE_RULE.COM asks you several questions regarding your local time zone and then creates a new UTC startup procedure, DTSS$UTC_STARTUP.COM. Execute the new UTC startup:

```
$ @SYS$STARTUP:DTSS$UTC_STARTUP
```

This process clears up the system time conflicts and you should be able to continue with your DCE configuration or startup operation.

14.2.2 Time Synchronization Problems

If your system clock is not synchronized with the system clock on the security server, you may receive an error during the HP DCE configuration. An error can occur even if the clocks are skewed by as little as five minutes.

Following is an example of an error that can occur because of a time synchronization problem between your system clock and the security server system clock:

```
    Please enter the principal name to be used [cell_admin]:
    Please enter the password for principal "cell_admin":

Establishing security environment for principal "cell_admin" . . .
Error: Cannot bind to the registry
Registry server unavailable (dce / sec) 249791450 (0x052000000)%SYSTEM-F-
                                                              ABORT,abort

***************************    ERROR    ***************************
***  An error occurred while setting up the security environment
***  using principal name "cell_admin"

Do you want to restart the client configuration (YES/NO/?) [Y]? n
```

A workaround is to set the time on your system to match the time on the node running the security server. On OpenVMS systems, use the following command:

```
$ SET TIME=dd-mmm-yyyy:hh:mm:ss
```

14.2.3 Time OPCOM Messages

Occasionally, OPCOM messages may appear on your screen. (These messages also are logged in SYS$MANAGER:OPERATOR.LOG.) You can safely ignore these messages as long as you have the DTS servers you need. (If you do not have the DTS servers you need, investigate the status of the DTS servers.)

Following are three messages you may see.

```
%%%%%%%%%%  OPCOM  27-SEP-1999 10:30:09.50  %%%%%%%%%%
Message from user SYSTEM on OPNDCE
dtsd.dce:  DCE error: Failure in rpc_mgmt_inq_server_princ_name:
/.../dceopnfst/hosts/opnvms/dts-entity communications
            failure (dce / rpc)
```

```
%%%%%%%%%%  OPCOM  27-SEP-1999 10:30:09.65  %%%%%%%%%%
Message from user SYSTEM on OPNDCE
dtsd.dce:  DCE error: Failure in rpc_mgmt_inq_server_princ_name:
/.../dceopnfst/hosts/opnvms/dts-entity not registered in
            endpoint map (dce / rpc)

%%%%%%%%%%  OPCOM  27-SEP-1999 13:46:04.70  %%%%%%%%%%
Message from user SYSTEM on OPNDCE
dtsd.dce:  DCE error: Error requesting time
            from server : communications failure (dce / rpc)
```

These messages indicate either the time daemon is not active because the system is down, you choose to have the time daemon stop running on a node, or the DTS daemon needs to be restarted because of an unexpected error.

14.3 Client/Server Problems

Successful DCE operation requires components on both the OpenVMS client system and your server system (for example, HP Tru64 UNIX Alpha) to work together. There are several things you can check on your client and on your server if DCE is not operating correctly.

14.3.1 OpenVMS Client System

To check the OpenVMS client system:

1. Run the CDS Control Program:

   ```
   $ RUN SYS$SYSTEM:DCE$DCECP
   dcecp> CELL SHOW /.:
   ```

   If DCE is working correctly, this will obtain cell information from the server and display it for you:

   ```
   {secservers
    /.../opndce-cell/subsys/dce/sec/master}
   {cdsservers
    /.../opndce-cell/hosts/opndce}
   {dtsservers
    /.../opndce-cell/hosts/opndce/dts-entity}
   {hosts
    /.../opndce-cell/hosts/opndce}
   ```

   If you get a socket error, a problem with communications within the local client system exists. Verify that HP TCP/IP Services for OpenVMS (UCX) is started on your system and ensure that it is configured for proper DCE operation.

2. Verify that the server system is reachable from your client system:

   ```
   $ UCX PING server_node
   ```

   If you get the following error:

   ```
    %UCX-E-GETHST, Error in getting host name
    %RMS-E-RNF, record not found
   ```

   then the server system host name is not defined in the UCX hosts database. You can define it in the database with the command:

   ```
   $ UCX SET HOST* hostname /ADDRESS=nn.nn.nn.nn
   ```

   Note that it is not required that your server hostname be defined in the local UCX hosts database for proper DCE operation. If your server host is not defined in the UCX hosts database, however, you will be asked to provide the Internet address of the server host during the DCE configuration process.

If the UCX PING command returns the message:

```
%UCX-I-LOOPACT, <SERVER_NODE>  is alive
```

then basic communication with your server node is working.

3. Next, ensure that the CDS library service is defined in the UCX services database:

   ```
   $ UCX SHOW SERVICE
   ```

   You should see a service definition for the service `cdsLib` in the listing, indicating the port number to be used for CDS client and clerk communication. Note that the service `state` should be `Disabled`. See the release notes for more information about the `cdsLib` service.

4. Examine the security PE_SITE file used to locate the security registry for the cell:

   ```
   $ TYPE DCE$SPECIFIC:[ETC.SECURITY]PE_SITE.;
   ```

   You will see text such as:

   ```
   /.../opndce-cell 535ace40-a138-11cc-ba08-08002b30910e@ncadg_ip_udp:16.32[]
   /.../opndce-cell 535ace40-a138-11cc-ba08-08002b30910e@ncacn_ip_tcp:16.32[]
   ```

   Compare this information with the output from the rpccp show mapping command on the server, as described in Section 14.3.2.

14.3.2 Server System

Note that the following examples assume a HP Tru64 UNIX DCE server system.

1. Use the DCESETUP SHOW option to ensure that the server daemons are active:

   ```
   # /etc/dcesetup show

   The following DCE daemons are active on this system:

     RPC daemon                    (rpcd)         pid: 756
     Security Server daemon        (secd)         pid: 762
     Security Client daemon        (sec_clientd)  pid: 768
     CDS Advertiser daemon         (cdsadv)       pid: 774
   ```

2. Ensure that CDS is functioning properly:

   ```
   # cdscp show dir /.:
   ```

   If this command fails or hangs, you may need to restart DCE on your server:

   ```
   # /etc/dcesetup restart
   ```

   If the `cdscp` command still fails, try a CLEAN operation followed by a START operation on the server:

   ```
   # /etc/dcesetup clean
   # /etc/dcesetup start
   ```

3. Ensure that the security registry is known to RPC:

   ```
   # rpccp show mapping
   ```

   You should see an object listed such as:

   ```
   <OBJECT>          2eef26c0-668f-11cc-8640-08002b35b39a
   <INTERFACE ID>    4c8782805000.0d.00.02.87.14.00.00.00,1.0
   <STRING BINDING>  ncacn_ip_tcp:25.0.0.145[1322]
   <ANNOTATION>      DCE user registry
   ```

Verify that the object UUID and the string binding protocol name and Internet address match the definitions in the PE_SITE file located on the OpenVMS DCE client system as described in Section 14.3.1. If they do not match, you must reconfigure the OpenVMS DCE client system.

14.4 Configuration and CDS

When DCE$SETUP starts, it may occasionally fail to contact the CDS master server. This may happen for one of the following reasons:

- Communication with the CDS server host fails or the CDS server process is not active on the server host.

  To correct this problem, make sure that DCE services are properly configured and started on the server system.

- The CDS advertiser on the local system has not seen a clearinghouse advertisement from the server. This problem may be caused because the CDS Server is not on the same LAN.

  To correct this problem, you need to configure the OpenVMS system and answer NO to the configuration question:

  ```
  Is the CDS Master Server within broadcast range (YES/NO/?) [Y]? NO
  ```

  You will then be asked to supply the hostname where the CDS master server is running. This causes CDS startup to use a CDSCP DEFINE CACHED SERVER command. If the server is available, this will force the server to send a clearinghouse advertisement to the client system Advertiser.

14.5 Configuration and Naming

When configuring a cell, you may receive an error similar to the following from `rpc_binding_set_auth_info()`:

```
 336760839 (decimal), 14129007 (hex):
 Server not found in Kerberos database (dce / krb)
```

To solve this problem, be sure that you do not configure a cell with the same name as another cell on the same network.

If you run system and functional tests that configure cells, make sure that the tests generate a unique name each time the test is run. You can also use the hostname of the server machine as part of the cell name.

14.6 Modifications to HP TCP/IP Services (UCX)

HP DCE for OpenVMS Alpha and OpenVMS I64 requires modification of several UCX parameters for proper operation. Make sure you read the current HP DCE for OpenVMS release notes for the most recent recommendations.

14.7 Principal Quota Exhausted

If you try to use DCE$RGY_EDIT to add a principal name, you may receive the following error message:

```
?(rgy_edit) Unable to add principal  "Xyzzy" - Principal quota
  exhausted (dce / sec)
```

The message means that your process does not have sufficient DCE credentials to complete the task. Therefore, you must login as `cell_admin` or another privileged DCE account before retrying the command.

14.8 Linking RPC Stub Modules into Shareable Images

If you build shareable images that contain RPC generated stub modules, you should use a linker options file. PSECT statements in the linker options file are used to resolve differences in the PSECT attributes between the RPC generated object file and the new shareable image. The following sections discuss how to solve problems that can arise when you create, link against, or activate a shareable image that contains RPC generated stub modules. This section can be summarized as follows:

- Program sections (PSECTs) in shareable images should be SHR,NOWRT or NOSHR,W RT unless the image is installed with privileges.
- Program sections in modules linked against shareable images must match exactly or conflicting PSECT errors will occur.
- Until the program runs, you may have to correct PSECT attributes as far back as the shareable image.

The PSECT attributes of the RPC generated interface specifications (IFspecs) should be set to the following:

```
(GBL,SHR,NOWRT)
```

RPC interface specifications usually do not change, so it is rarely required that they be set to a writable PSECT attribute. RPC interface specifications are frequently shared. If your shareable image contains more than one cluster and the same interface specification is defined in multiple object modules, these interface specifications can be effectively collected into the same global cluster with the GBL PSECT attribute. Note that, in this case, the first module encountered by the linker that defines the IFspec will be used to initialize the value of the IFspec in the shareable image. A map file can help you identify and correct problems with PSECTs and their contents. The contents of any PSECT should be nonzero.

If you find a zero byte PSECT, you may need to explicitly specify the module name in the options file. The module name can be specified directly on its own or as part of the `/library/include=()` statement associated with an object library. PSECTs should not be zero unless they are initialized at runtime, and this presumes that the PSECT is writable (WRT).

14.8.1 Errors Creating a Shareable Image

The following examples show some of the errors that might occur when you try to create a shareable image with RPC stub object modules:

```
$ link/share/exe=myshr.exe/map=myshr.map -
$-        test1_mgr,test1_sstub,dce:dce.opt/opt
$ %LINK-I-BASDUERRS, basing image due to errors in relocatable
references
$ %LINK-W-ADRWRTDAT, address data in shareable writeable section
$ in psect TEST1_V0_0_S_IFSPEC offset %X00000000
$ in module TEST1_SSTUB file USER:[MY.CODE.DCE]TEST1_SSTUB.OBJ;
$
```

The PSECT name is causing the linker problem. To correct this problem, create an option file including the following line, and place it on your link command line as follows:

```
$ create myopt.opt
$ PSECT= TEST1_V0_0_S_IFSPEC, shr,nowrt,gbl
$ ctrl-z
$
$ link/share/exe=myshr.exe/map=myshr.map -
$-        test1_mgr,test1_sstub,dce:dce.opt/opt,myopt.opt/opt
```

This will remove the link problems so that you can create a shareable image. There are still errors in this shareable image whose solutions are shown in the following examples.

14.8.2 Errors Linking Against a Shareable Image

Once you have a shareable image, you may still see linker problems related to the PSECT attributes between the shareable image and new object files. In the following example, a main routine is linked against the same shareable image from the previous example. The new object module references some of the same variables defined by the RPC stub module.

```
$ link/exec=test1d/map=test1d.map test1_main,sys$input/opt
$ myshr.exe/share
$ ctrl-z
$
$ %LINK-W-MULPSC, conflicting attributes for psect TEST1_V0_0_S_IFSPEC
$       in module TEST1_MAIN file USER:[MY.CODE.DCE]TEST1_MAIN.OBJ;
$
```

If you search the map files of both `myshr.map` and `test1d.map` for the PSECT TEST1_V0_0_S_IFSPEC, you will see that the PSECT attributes for this PSECT match; however, the map files are incorrect. The solution to this link problem is to include the PSECT directive in a linker options file for the offending PSECT name. The previous example simply typed in the options from the command line, but you should place these linker statements in a linker option file. The options are typed in from SYS$INPUT in the following example:

```
$ link/exec=test1d/map=test1d.map test1_main,sys$input/opt
$ PSECT= TEST1_V0_0_S_IFSPEC, shr,nowrt,gbl
$ myshr.exe/share
$ ctrl-z
$
```

14.8.3 Errors Activating Shareable Images

When you run this program, the following results occur:

```
$ run test1d
$ %DCL-W-ACTIMAGE, error activating image MYSHR
$ -CLI-E-IMAGEFNF, image file not found SYS$LIBRARY:MYSHR.EXE
$
```

To allow the image activator to check a directory other than SYS$LIBRARY for your new shareable image, you must define a logical name or you must copy your new shareable image into SYS$LIBRARY. In the following example, a logical name is defined and the program is run again with the following results:

```
$ define MYSHR sys$disk:[]myshr.exe;
$
$ run test1d
$ %DCL-W-ACTIMAGE, error activating image MYSHR
$ -CLI-E-IMGNAME, image file USER:[MY.CODE.DCE]MYSHR.EXE;
$ -SYSTEM-F-NOTINSTALL, writable shareable images must be installed
$
```

The problem is in the myshr.exe image: myshr.exe has PSECTs whose PSECT attributes specify *both* SHR and WRT. The solution is to add the correct PSECT attributes to the offending PSECTs in the myshr.exe shareable image to myshr.opt. This can be done on the command line, as follows:

```
$ link/share/exe=myshr.exe/map=myshr.map -
    test1_mgr,test1_sstub,dce:dce.opt/opt,sys$input/opt
    psect= TEST1_V0_0_S_IFSPEC, shr,nowrt,gbl
    psect= RPC_SS_ALLOCATE_IS_SET_UP, noshr,wrt,gbl
    psect= RPC_SS_CONTEXT_IS_SET_UP, noshr,wrt,gbl
    psect= RPC_SS_SERVER_IS_SET_UP, noshr,wrt,gbl
    psect= RPC_SS_THREAD_SUPP_KEY, noshr,wrt,gbl
    psect= RPC_SS_CONTEXT_TABLE_MUTEX,noshr,wrt,gbl
    psect= TEST1_V0_0_C_IFSPEC, shr,nowrt,gbl
    ctrl-z
$
```

All of the PSECTs that existed in the myshr.map mapfile that had SHR and WRT attributes were changed so that the PSECT was either SHR,NOWRT or NOSHR,WRT. The choice depends upon your use of the data item. IFspecs are usually shared and non-writable. The RPC_SS PSECTs are written and not generally shared among program images linked against the shareable image.

The following example tries to relink the main program again, but another problem occurs:

```
$ link/exec=test1d/map=test1d.map test1_main,sys$input/opt
$ PSECT= TEST1_V0_0_S_IFSPEC, shr,nowrt,gbl
$ myshr.exe/share
$ ctrl-z

$ %LINK-W-MULPSC, conflicting attributes for psect TEST1_V0_0_C_IFSPEC
  in module TEST1_MAIN file USERE:[MY.CODE.DCE]TEST1_MAIN.OBJ
```

Because the PSECT attributes of the TEST1_V0_0_S_IFSPEC PSECT was changed in the shareable image, its reference in test1_main.obj is not correct. To solve this problem, add the correct PSECT attribute. For example:

```
$ link/exec=test1d/map=test1d.map test1_main,sys$input/opt
$ PSECT= TEST1_V0_0_S_IFSPEC, shr,nowrt,gbl
$ PSECT= TEST1_V0_0_C_IFSPEC, shr,nowrt,gbl
 myshr.exe/share
 ctrl-z

$
```

In the final example, the **test1d** program is run and the desired results occur:

```
$ run test1d
 ncacn_ip_tcp 16.32.0.87 3314\bold)
 ncacn_dnet_nsp 63.503 RPC270002590001\bold)
 ncadg_ip_udp 16.32.0.87 1485
```

14.9 Integrated Login Problems

The following sections describe problems that may occur when Integrated Login is enabled on your system, and solutions to those problems.

14.9.1 No Logical Name Match Error When Integrated Login Is Enabled

If you receive the error:

```
\*C%SYSTEM-F-NOLOGNAM, no logical name match)
```

when you try to set host to another system, the problem may be occurring because the SYSGEN parameter LGI_CALLOUTS has been set nonzero, but the logical name LGI$LOGINOUT_CALLOUTS has not been defined.

This situation can only occur as a result of one of the following:

- An incomplete startup or shutdown of DCE
- A privileged user altered the SYSGEN parameter LGI_CALLOUTS or the logical name LGI$LOGINOUT_CALLOUTS

To solve this problem, enter one of the following commands to reenable Integrated Login by running DCE$SETUP or DCE$STARTUP, as follows:

```
$ @sys$startup:dce$setup start
$ @sys$startup:dce$startup
```

14.9.2 Potential Integrated Login and SYSGEN Problems

The Integrated Login component of DCE uses the SYSGEN parameter LGI_ CALLOUTS. LGI_CALLOUTS must be set to 1 only in the ACTIVE SYSGEN parameter set when DCE is running with Integrated Login enabled. LGI_ CALLOUTS must never be set to 1 in the CURRENT SYSGEN parameter set — this would prevent all logins from occurring on a subsequent reboot of the system. See the chapter on Integrated Login for more information.

The following paragraphs explain how to solve this problem if it occurs.

If you cannot log in because LGI_CALLOUTS is set to 1 and DCE is not running, there are two solutions, as follows:

- If you are already logged in to the system, use SYSGEN to correct the problem.

  ```
  $ RUN SYS$SYSTEM:SYSGEN
  SYSGEN> SET LGI_CALLOUTS 0
  SYSGEN> WRITE ACTIVE
  SYSGEN> EXIT
  $
  ```

- Reboot the system with a conversational boot and ensure that the LGI_ CALLOUTS parameter is zero.

  ```
  SYSBOOT> SET LGI_CALLOUTS 0
  SYSBOOT> C
  ```

15
Example Programs

The Application Developer's Kit of HP DCE for OpenVMS Alpha and OpenVMS I64 contains 17 example programs. These programs are located in subdirectories of the following directory:

SYS$COMMON:[SYSHLP.EXAMPLES.DCE]

The examples demonstrate some of the basic capabilities of DCE as well as the steps required when writing DCE distributed applications. All of the example programs are written in HP C or HP C++ except the Payroll example, which is written in HP Fortran.

Each example is in a separate directory and contains the following files needed to build and run the example:

- Source files for the example
- <example>.readme file, which explains the steps required to build and run the program
- <example>.com file, which compiles and links the program

The example programs vary in complexity. The RPC Test Programs and Hello_svc programs are elementary; the Generic_app and Timop_svc programs are more complex. The example programs illustrate some of the DCE interfaces and services.

Table 15–1 briefly describes each example program located in subdirectories of SYS$COMMON:[SYSHLP.EXAMPLES.DCE]:

Table 15–1 Example Program Features

Example Program	Description
[.GENERIC_APP]	This sample application illustrates the recommended procedures for writing DCE distributed applications. The code is as generic as possible, and demonstrates what most servers need to do. The application is mostly initialization and cleanup code, with extensive examples of ACL management, serviceability code, security setup, and signal handling.
[.GSSAPI]	The ECHO application demonstrates how a distributed application secures itself using the GSSAPI security interface.
[.PWD_MGMT]	This example provides a sample application for the password management server, a cell-wide service for enforcing the password selection policy users must follow when updating their passwords.

(continued on next page)

Table 15–1 (Cont.) Example Program Features

Example Program	Description
[.RPC.BOOK]	The Distributed Calendar (BOOK) program is a fairly sophisticated client/server application that uses several DCE services. The client sets up a search context that imports a binding handle for a book server from the directory service.
[.CONTEXT_APP]	The context_app server maintains a storage area that the client will write to and read from. The client accesses the storage area using a context handle obtained from the server.
[.RPC.DATA_TEST_APP]	This example demonstrates the use of various RPC data types.
[.RPC.PAYROLL]	The payroll software is a simple client/server application that makes minimal use of the DCE services. Its purpose is to show a complete example application with FORTRAN.
[.RPC.PHONEBOOK]	RPC PHNBK is an application that distributes a phone number directory. The server registers transport endpoints with the RPC endpoint mapper, and binding information is exported to the directory service.
[.RPC.TEST1]	RPC Test Program #1 (TEST1) is a very simple client /server program that makes minimal use of the DCE services. It is useful for acquiring the basics of client /server programming. The server does not register transport endpoints with the RPC daemon, and no binding information is exported to the directory service. The user has to manually transfer the server binding information to the client.
[.RPC.TEST2]	RPC Test Program #2 is a simple client/server program that makes slightly more use of the DCE services than does the RPC Test Program #1. In this program, the server registers transport endpoints with the RPC daemon, and exports binding information to the directory service. The client uses the auto-handle mechanism to import server binding information.
[.RPC.TEST3]	RPC Test Program #3 is a simple client/server program that makes minimal use of the DCE services. The server does not register transport endpoints with the RPC daemon, and no binding information is exported to the directory service. The user has to manually transfer the server binding information to the client.
[.RPC.IDLCXX.ACCOUNT]	This C++ example tests inheritance, binding to an object using another interface, binding to an object with an unsupported interface, and the reflexive, symmetric, and transitive relation properties of the bind( ) API.
[.RPC.IDLCXX.ACCOUNTC]	This C++ example tests the same properties as the account example, but uses the C interfaces for all the APIs.
[.RPC.IDLCXX.CARD]	This C++ example tests the passing of C++ objects as parameters using the `[cxx_delegate]` attribute and the polymorphism property of the base class.

(continued on next page)

Table 15–1 (Cont.) Example Program Features

Example Program	Description
[.RPC.IDLCXX.STACK]	This C++ example tests the passing of C++ objects as parameters using the `[cxx_delegate]` attribute and a user-defined stack class.
[.SVC.HELLO_SVC]	This is the Hello World example program for the DCE Serviceability API, a simple illustration of the new serviceability functionality.
[.SVC.TIMOP_SVC]	The timop program is a tutorial DCE application example. It exercises the basic DCE technologies: threads, RPC, security, directory, time and serviceability.

Please see the HP DCE for OpenVMS Alpha and OpenVMS I64 release notes for current restrictions on using the example programs.

A

Using NSedit

This appendix provides information on using the Namespace Editor (NSedit). Note that at this time NSedit is only a prototype of a system management tool. Note also that this release of NSedit does not yet have complete functionality.

NSedit is a graphical user interface to the namespace. A namespace is a collection of names that one or more CDS servers know about, look up, modify, and share. Usually only one namespace is associated with a particular cell.

A.1 Starting NSedit

To start NSedit, you must first log in to DCE as `cell_admin`. Then, enter the following command:

```
$ MCR DCE$NSEDIT
```

A.2 NSedit Functionality

NSedit is a graphical user interface for CDS. Although it does not provide the complete functionality of the CDS clerk interface, it does provide a user-friendly environment for creating, viewing, and modifying entries in a namespace.

NSedit provides an additional level of namespace data caching. The NSedit cache is used to store data read from the clerk's local cache or from the CDS server. This allows for faster retrieval of data at the expense of accuracy. (Changes may have occurred in the namespace that are not reflected in the NSedit cache.) The tradeoff between speed and accuracy can be controlled by setting the appropriate mode in the Display menu of the tree browser.

NSedit consists of three windows:

- Tree Browser Window

  The entire left window of the NSedit screen is a tree browser that allows users to view the hierarchical structure of the namespace and to create and delete entries.

- Entry Attributes Window

  The top right window is a list management utility that allows users to view, modify, and delete attributes and values of namespace entries.

  Note that most of the features in this window are Read Only for this release.

- ACL Window

  The bottom right window is a second tree browser that allows users to view, modify, and delete ACLs of namespace entries.

A.2.1 Tree Browser Window

The tree browser lets you view the namespace and create and delete namespace entries. Each CDS entry is represented in the tree browser by a rectangle containing the name of the entry and an icon indicating the entry type. A CDS entry can be one of the following:

- Directory: A directory contains objects, soft links, or other directories (child pointers) as its children. The icon is a filled square.
- Object: An object represents a physical resource in the network. Its icon is an empty square.
- Group: A group is a set of names in the network. The icon is a set of seven small squares.
- Soft link: A soft link is a pointer to a CDS entry. The icon is a right arrow.

The tree browser provides the user with the ability to expand a node and view all its children (objects, subdirectories, or soft links) by clicking Mouse Button 3 (MB3) on a tree node or by specifying the appropriate maximum tree level to display, as described in Section A.3.1.

A.2.2 Entry Attributes Window

The Entry Attributes window is the part of NSedit that lets you create, view, modify, and delete attributes and values from an existing object, directory or soft link. When you select a CDS entry with the tree browser, the attributes and values of this entry are shown on the Entry Attributes window. You can then add, modify, or delete attributes from this entry, as described in this appendix.

A.2.3 ACL Window

This window lets you perform the following functions.

- List the ACL entries.
- Add an ACL entry.
- Modify an ACL entry.
- Delete an ACL entry.
- Substitute all ACL entries with a new ACL entry. This function is not yet implemented.
- Load an ACL from a file. This function is not yet implemented.
- Save an ACL to a file. This function is not yet implemented.
- Copy an ACL entry to another. This function is not yet implemented.
- Copy an ACL to another (for example, an Object ACL can be copied onto an Initial Container Creation ACL for the selected object). This function is not yet implemented.
- Delete all ACL entries except the `user_obj` entry (kill the ACL). This function is not yet implemented.
- List the available permission tokens.

A.3 Common Uses of NSedit

The following sections describe what you can do with NSedit.

A.3.1 Expanding and Collapsing Tree Nodes

To expand a tree node and view its children, click MB3 on the node. This works only for directory and soft link nodes. For example, by successively expanding nodes, you can see all of the host's children as well as their children.

To collapse a tree node whose children are at a higher level than the maximum tree level specified by the user (with the Set Level selection of the Display menu), click MB3 on the node. The node will not collapse if the level of its children is lower than the maximum tree level. The default maximum tree level is 1, which means that only the children of the root node are displayed.

A.3.2 Creating an Object or a Directory

To create an object or directory:

1. Type in the name of the new object or directory in the top left selection area and press RETURN. The `create_entry_dialogue_popup` appears.
2. Select Object, Directory, Group, or Soft Link and press OK.

If the object or directory has been successfully created, a new rectangle with the name of the new object or directory appears as a child of the specified parent directory. If CDS cannot create the object or directory, an error message appears.

A.3.3 Creating and Viewing a Soft Link

Note that the following functionality is not yet supported.

1. Click the left mouse button on a rectangle, to define the target of the soft link to be created. This target entry appears highlighted.
2. Type in the name of the new soft link in the prompt window and press RETURN.

If the soft link has been successfully created, a new rectangle with the name of the new soft link is displayed as a child of the specified parent directory. To verify the creation of the soft link, you can click the right mouse button on the soft link rectangle. If CDS could not create the soft link, an error message appears.

A.3.4 Deleting an Entry

To delete an entry:

1. Click the left mouse button on a rectangle, to define the target to be deleted. This target entry is displayed highlighted.
2. Select Delete from the Edit menu.

If the entry has been successfully deleted, the rectangle will also disappear from the tree structure. If CDS could not delete the entry, an error message is displayed.

A.3.5 Viewing Attributes and Values

To view the attributes and values of a CDS entry, click the left mouse button on this entry in the Tree Browser window. The attributes and values of this entry are displayed in the Entry Attributes window. An alternative way to view an entry is to type its full name in the top left selection area. Then the attributes and values of this entry are displayed in the Entry Attributes window (or an empty list will appear if the entry does not exist).

A.3.6 Creating a Group and Adding Members

To create a group, follow the steps described in Section A.3.2. Select Create Group from the `create_entry_dialog_popup`.

A.4 NSedit Menus and Dialog Box

There are three pull-down menus in NSedit and one popup dialog box:

- File Menu
- Display Menu
- Edit Menu
- Create Entry Pop-up Dialog Box

A.4.1 File Menu

The File menu has the following choice:

- Exit: This selection exits NSedit.

A.4.2 Display Menu

The Display menu has the following choices:

- Set Root: This selection allows the user to specify the root of the tree structure to display. After a root is specified, only entries that are children of the root will be displayed by the tree browser.
- Set Depth: This selection allows the user to specify a maximum tree depth to display. Nodes deeper than the specified depth are not displayed, unless explicitly selected for expansion (by clicking the right mouse button).
- Show Entry: This selection toggles between enabling and disabling the Entry Attributes window. When the Entry Attributes window is disabled, no text can be typed in its text input areas and these areas as well as the area where attributes and values appear grayed.
- Show ACLs: This selection toggles between enabling and disabling the ACL window. When the ACL window is disabled, no text can be typed into its text input area. These areas, as well as the area where ACLs appear, are grayed. The ACLs menu selections also are grayed to indicate that no operations are possible at that time.
- Caching: This menu sets the level of caching for NSedit operations:
- Internal: This selection tells the program to read the children of a directory from the program's own cache and not from the namespace. Every time a directory's children are read from the namespace, they are placed in a local cache (data structure) by NSedit. In Internal mode, every time a node is collapsed and reexpanded, its children are read from this cache. You want to use this mode most of the time, unless you suspect that new entries

have been created in the namespace. In that case you have to collapse the directory entry, select either Local mode or Server mode, and click the right mouse button on the directory to reexpand the directory and see the new information.

- Local: This selection tells the program to read the children of a directory from the CDS local cache (as opposed to reading them from the program's own cache or querying the CDS server) when you click the right mouse button on a directory entry. This selection is useful when new information is stored in the CDS local cache but NSedit is not aware of it (because it has already stored some old information in its own cache).
- Server: This selection tells the program to query the CDS server when reading children of a directory (as opposed to reading them from the program's own cache or from the CDS local cache) when you click the right mouse button on a directory entry. This selection is useful when new information has been stored in the namespace (and the server is aware of it), but is not yet propagated to the CDS local cache (as, for example, when another user creates a new object on a different machine).

A.4.3 Edit Menu

The Edit menu has the following choices:

- Delete Entry: Deletes a selected entry. Select an entry in the Tree Browser window, pull down the Edit menu, and choose Delete Entry to delete the selected entry.
- Delete Attribute: This selection sets the delete mode for attribute names.
- Delete Value: This selection sets the delete mode for attribute values.
- Delete: Deletes an ACL entry or a complete ACL.
- Substitute ACL: (Not supported in this release.) Substitutes all ACL entries with a new ACL entry.
- Kill ACL: (Not supported in this release.) Removes all ACL entries other than type `user_obj` from a given ACL.
- Show Tokens: (Not supported in this release.) Shows the tokens allowed by the ACL manager for permission strings.

A.4.4 Create Entry Pop-up Dialog

The Create Entry Pop-up Dialog box has the following choices:

- Create Object: This selection sets the create object mode. Select OK in the dialog box to create a new object as a child of the directory node.
- Create Directory: This selection sets the create directory mode. Select OK in the dialog box to create the new (sub)directory as a child of the directory node.
- Create Soft Link: (Not supported in this release.) This selection sets the create soft link mode. To create a soft link, you click the left mouse button on the target node and type in the full name of the soft link in the prompt window.
- Create Group: This selection sets the create group mode. Select OK in the dialog box to create a new group as a child of the directory node.

Index

F

H

I

L

M

N

O

P

R

S

T

V

W