IBM Distributed Computing Environment Version 3.1
for AIX and Solaris:

Introduction to DCE

IBM Distributed Computing Environment Version 3.1
for AIX and Solaris:

Introduction to DCE

Note
Before using this document, read the general information under “Appendix. Notices” on page 121.

First Edition (August, 1999)

This edition applies to Version 3.1 of *IBM Distributed Computing Environment for AIX and Solaris* and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. Send your comments to the following address:

International Business Machines Corporation

Department VLXA

11400 Burnet Road

Austin, Texas

78758

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

Contents

Preface

IBM DCE Version 3.1 for AIX and Solaris: Introduction to DCE provides an introduction to the IBM® Distributed Computing Environment (DCE) offering. The glossary introduces terms used in DCE documentation.

Audience

The content and intended audience of this manual change from less technical to more technical as the manual progresses. “Chapter 1. Overview of DCE” on page 1 is written for anyone interested in an overview of DCE, including managers, system administrators, and application programmers. “Chapter 2. DCE Configuration” on page 17 is intended for network managers and administrators. “Chapter 3. DCE Technology Components” on page 25 and “Chapter 4. Integration of DCE Technology Components” on page 77 are targeted primarily for administrators and programmers.

“Overview of DCE Documentation” on page 81 is written for anyone wishing to find further information on DCE. It suggests reading paths through the DCE documentation set for various audiences. The “Glossary” on page 85 contains terms used throughout the DCE documentation. Each term is defined for the audience of the manual in which it appears. For example, the definition of a term used in the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Core Components* is targeted for the same audience as the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Core Components* itself.

Applicability

This revision applies to the IBM DCE for AIX® and Solaris, Release 3.1 offering and related updates.

Purpose

After reading this document, a user will

1. Have a high-level understanding of DCE
2. Understand the individual technology components that constitute DCE
3. Understand the interdependencies of the DCE technology components
4. Be able to find further information about DCE in related documents

Document Usage

The manual is organized as follows:

“Chapter 1. Overview of DCE” on page 1

Gives an overview of DCE. It describes distributed computing and its uses, and presents the client/server model of distributed computing, on which DCE is based. It gives a summary of the DCE architecture, along with a brief description of each of the technology components that make up DCE, and their integration with one another.

Related Documents

The DCE documentation set comprises the following manuals. See Appendix A for a description of each of these manuals.

- *IBM DCE Version 3.1 for AIX and Solaris: Introduction to DCE*
- *IBM DCE Version 3.1 for AIX: Quick Beginnings*
- *IBM DCE Version 3.1 for Solaris: Quick Beginnings*
- *IBM DCE Version 3.1 for AIX and Solaris: Administration Commands Reference*
- *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Introduction*
- *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Core Components*
- *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Introduction and Style Guide*
- *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Core Components*
- *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Directory Services*

- *IBM DCE Version 3.1 for AIX and Solaris: Application Development Reference*
- *IBM DCE Version 3.1 for AIX and Solaris: Problem Determination Guide OSF DCE Technical Supplement*
- *IBM DCE Version 3.1 for AIX: Release Notes*
- *IBM DCE Version 3.1 for Solaris: Release Notes*
- *IBM DCE Version 3.1 for AIX: High Availability Cluster Multi-Processing Guide for DCE and DFS*

Typographic and Keying Conventions

This guide uses the following typographic conventions:

Bold **Bold** words or characters represent system elements that you must use literally, such as commands, options, and pathnames.

Italic *Italic* words or characters represent variable values that you must supply. *Italic* type is also used to introduce a new DCE term.

Constant width
Examples and information that the system displays appear in `constant width` typeface.

[ ] Brackets enclose optional items in format and syntax descriptions.

{ } Braces enclose a list from which you must choose an item in format and syntax descriptions.

| A vertical bar separates items in a list of choices.

< > Angle brackets enclose the name of a key on the keyboard.

... Horizontal ellipsis points indicate that you can repeat the preceding item one or more times.

dcelocal
The OSF variable *dcelocal* in this document equates to the AIX variable **/opt/dcelocal**.

dceshare
The OSF variable *dceshare* in this document equates to the AIX variable **/opt/dcelocal**.

This guide uses the following keying conventions:

<Ctrl- *x***>** **or** ^ *x*
The notation **<Ctrl-** *x***>** or ^ *x* followed by the name of a key indicates a control character sequence. For example, **<Ctrl-C>** means that you hold down the control key while pressing **<C>**.

<Return>
The notation **<Return>** refers to the key on your terminal or workstation that is labeled with the word Return or Enter, or with a left arrow.

Chapter 1. Overview of DCE

Unless otherwise stated, substitute International Business Machines (IBM) for all occurrences of the Open Software Foundation (OSF) made in this publication.

In this publication, IBM Distributed Computing Environment for AIX and Solaris, Version 3.1 is referred to as IBM DCE 3.1.

IBM's Distributed Computing Environment (DCE) provides services and tools that support the creation, use, and maintenance of distributed applications in a heterogeneous computing environment. This chapter provides an overview of DCE, beginning with a section describing distributed computing and its benefits. The next section describes three distributed computing models—client/server, remote procedure call (RPC), and data sharing. The final section gives an overview of DCE itself, describing its technology components, the organization of a DCE environment, and the relationship between DCE and the underlying computing system.

Why Distributed Computing?

By "distributed computing" we mean computing that involves the cooperation of two or more machines communicating over a network (see Figure 1). The machines participating in the system can range from personal computers to supercomputers; the network can connect machines in one building or on different continents.

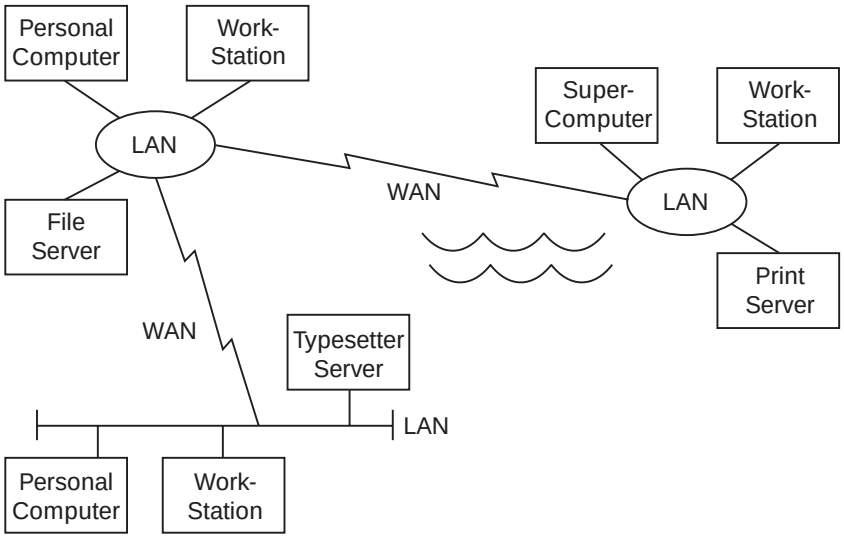

Figure 1. A Potential DCE Network

Why is enabling this type of cooperative computing important? One reason is historical: computing resources that used to operate independently now need to work together. For example, consider an office that acquired personal workstations for individual use. After a while, there were many workstations in the office building, and the users recognized that it would be desirable to share data and resources among the individual computers. They accomplished this by connecting the workstations over a network.

A second reason is functional: if there is special-function hardware or software available over the network, then that functionality does not have to be duplicated on every computer system (or *node*) that needs to access the special-purpose resource. For example, an organization could make a typesetting service available over the network, allowing users throughout the organization to submit their jobs to be typeset.

A third reason is economical: it may be more cost effective to have many small computers working together than one large computer of equivalent power. In addition, having many units connected to a network is the more flexible configuration, because if more resources are needed, another unit can be added in place, rather than bringing the whole system down and replacing it with an upgraded one.

Finally, a distributed system can be more reliable and available than a centralized system. This is a result of the ability to replicate both data and functionality. For example, when a given file is copied on two different machines, then even if one machine is unavailable, the file can still be accessed on the other machine. Likewise, if several printers are attached to a network, then even if an administrator takes one printer offline for maintenance, users can still print their files by using an alternate printer.

Distributed computing inherently brings with it not only potential advantages, but also new problems. Examples are keeping multiple copies of data consistent, and keeping the clocks on different machines in the system synchronized. A system that provides distributed computing support must address these new issues.

Why DCE?

Given that, for one of the reasons previously mentioned or some other reason, an organization decides that it wants to acquire distributed computing capability, why is DCE in particular advantageous? Why would an organization with a network such as the one in Figure 1 on page 1 benefit from using DCE to enable distributed computing?

DCE's benefits can be categorized into its support of distributed applications, the integration of its components with each other, DCE's relationship to its platforms, its support for data sharing, and DCE's interaction with the world outside of DCE, as described in the following list.

1. DCE provides tools and services that support distributed applications.

   DCE provides a high-level, coherent environment for developing and running applications on a distributed system. The DCE components fall into two categories: tools for developing distributed applications, and services for running distributed applications. The tools, such as DCE RPC and DCE Threads, assist in the development of an application. The services, such as the DCE Directory Service, Security Service, and Distributed Time Service, provide the support required in a distributed system that is analogous to the support an operating system provides in a centralized system.

   It is *possible* to develop distributed applications with much less assistance than what DCE offers. Programmers can write applications that cooperate across machines by explicitly writing the code that performs the network communications between them, but this requires much time and expertise. Programmers can also write distributed applications by using a communications tool, such as remote procedure call, while explicitly using

other necessary technologies, like standalone name and security services. However, DCE provides a set of components necessary for distributed computing that are already integrated, and that do as much work as possible automatically for the application programmer, system administrator, and end user.

2. DCE's set of services is integrated and comprehensive.

   A second benefit is the integration and comprehensiveness of the DCE components. Not only does DCE provide all the tools and services needed for developing and running distributed applications, but the DCE components themselves are well integrated. They use one another's services whenever possible, since many of the DCE components are themselves distributed applications. In addition to supporting the development of distributed applications, DCE includes services that address some of the new problems inherent in the distributed system itself, such as data consistency and clock synchronization. Finally, DCE includes management tools for administering all of the DCE services and many aspects of the distributed environment itself.

3. DCE provides interoperability and portability across heterogeneous platforms.

   A benefit of DCE is its orientation toward heterogeneous rather than homogeneous systems. One way to implement a distributed system is to use a single operating system that runs on all nodes participating in the distributed network. The DCE architecture, however, allows for different operating systems and hardware platforms. Using DCE, a process running on one computer can interoperate with a process on a second computer, even when the two computers have different hardware or operating systems. DCE can therefore accommodate a wider range of networks—especially networks needing distributed computing for the historical reasons previously listed—than a model that requires the same operating system running on every node. Applications that are built using DCE are portable to other hardware/operating system platforms that run DCE.

4. DCE supports data sharing.

   Another benefit is DCE's support of data sharing through its directory service and distributed file service. A user anywhere in the distributed system can share data by placing it in the namespace or in a file, whichever is appropriate for the application. The data is then accessible by authorized users throughout the system.

5. DCE participates in a global computing environment.

   One final benefit of DCE is the way it interacts with the outside world. In addition to supporting cooperation within and between themselves, DCE systems can also interoperate with computing environments outside of DCE. In particular, the DCE Directory Service can interoperate with two standard, global directory services—X.500 and Domain Name Service (DNS)—allowing users from within DCE to access information about the outside world. In this way, DCE participates in a global directory service. One benefit of such participation can be seen in DCE's distributed file system: it looks like one global file system, and users anywhere in the world can address the same file by using the same global name.

Potential Users of DCE

This section gives some examples of computing environments that can profit from distributed computing capabilities. In general, any computing organization wishing to take advantage of the benefits of a distributed computing environment—data and resource sharing, extensibility, availability, interoperability—can benefit from using DCE. For example:

1. An office with isolated computing resources can network the computers together and use DCE for data and resource sharing.
2. An organization consisting of multiple computing sites that are already interconnected by a network can use DCE to tie together and access resources across the different sites. The different sites can be in different countries, or even on different continents.
3. Any computing organization comprising, or expecting to comprise in the future, more cooperating hosts than can be easily administered manually (perhaps over a dozen nodes) can benefit greatly from the administrative support afforded by a DCE environment. For example, in DCE the database of computer users and their associated information (such as passwords) can be administered centrally, removing the need for an administrator to update information on every single node in the network each time a new user is added.
4. Organizations that write distributed applications can use DCE as a platform for their software. Applications that are written on DCE can be readily ported to other software and hardware platforms that also support DCE.
5. Organizations wishing to use applications that run on DCE platforms.
6. Organizations that wish to participate in networked computing on a global basis. Since DCE supports standard directory services that will be used throughout the world, a site that participates in DCE will be able to plug into that worldwide directory service database, allowing it to both ″see'' and access information about other sites and organizations around the world. In turn, it will be able to add itself to the directory service, allowing itself to be ″seen'' and accessed, if desired, by other sites worldwide.
7. System vendors whose customers are in any of the preceding categories.
8. Organizations that would like to make a service available over the network on one system (for example, a system running other than a UNIX® operating system), and have it accessible from other kinds of systems (for example, workstations running UNIX).

Models of Distributed Computing

DCE is based on three distributed computing models—client/server, remote procedure call, and data sharing. The client/server model is a way of organizing a distributed application. The remote procedure call model is a way of communicating between parts of a distributed application. The data sharing model is a way of handling data in a distributed system.

DCE also supports a distributed object model, which is a way of distributing data and functionality together in neat application packages known as distributed objects.

The following subsections briefly describe each model.

The Client/Server Model

A useful model for implementing distributed applications is the *client/server* model. In this model, the distributed application is divided into two parts, one part residing on each of the two computers that will be communicating during the distributed computation (see Figure 2 on page 5).

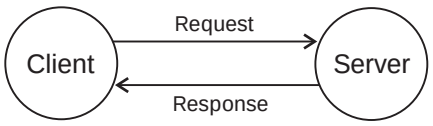

Figure 2. The Client/Server Model

The *client* side of the application is the part that resides on the node that initiates the distributed request and receives the benefit of the service (for example, a workstation that requests that a file be printed). The *server* side of the application is the part that resides on the node that receives and executes the distributed request (for example, the node with the printer). In this model, two different sets of code are produced—one that runs as a client, the other as a server.

Figure 3 shows a workstation running the client side of a distributed print program, and a print server running the server side of the distributed program.

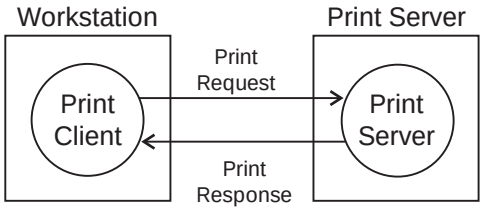

Figure 3. Communication Between the Print Client and Print Server

Note that the terms *client* and *server* can be seen as relative roles rather than as absolutes. For example, in executing the print request, the print server may in turn become a client in a distributed communication; that is, it may ask the file server to send it a copy of the file to be printed (see Figure 4).

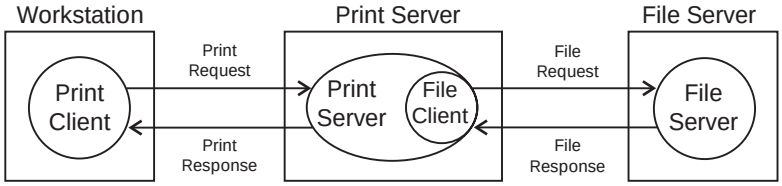

Figure 4. The Print Server Acting as a Client of the File Server

The terms *client* and *server* are also used to refer to specific nodes. This can be confusing since a given node, or even a given process, can be acting in both the client and server role. Nevertheless, it is often convenient to use the term file server when referring to the node on which the server side of a distributed file system is running—probably a machine that contains a lot of disk storage. Likewise, the *directory server* is a node that contains a database with names in it, and answers requests for access to those names. When clarification is needed, we use the term *machine* to indicate the node rather than the role. For example, in Figure 4, the print server, which runs on the print server machine, is acting as a client to the file server.

Note that it is possible for more than one server to run on a given node. For example, both a security server and a time server can run on the same machine. In

this case, the given node is both the security server machine and the time server machine (see Figure 5).

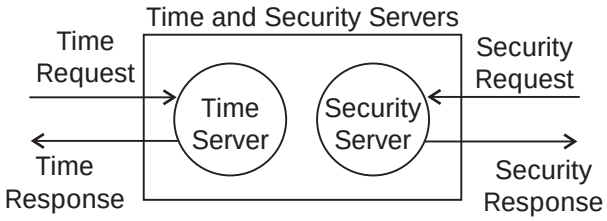

Figure 5. Two Servers Running on One Node

In general, when referring to clients and servers as nodes, the server nodes are specialized—they require software that is found only on that particular server (for example, the directory server); whereas client nodes are generalized—client machines are typically configured with the capability to be many types of client (for example, a directory, file, and security service client). See Figure 6.

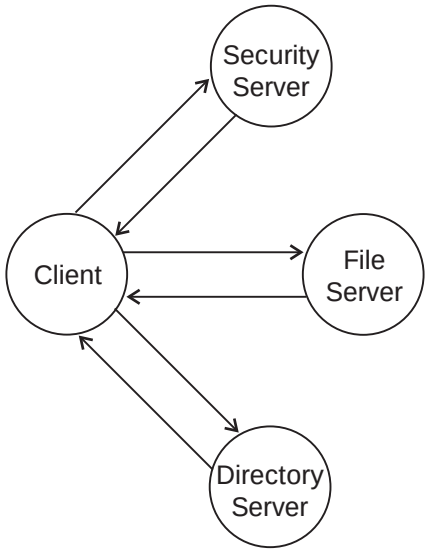

Figure 6. A Client is General; Servers are Specialized

The reason client nodes are generalized is that the client code is usually relatively small compared to the code that implements a server, and typically many nodes need to be able to run the client side of an application; whereas only one or two nodes may be equipped to run the server side of an application.

One final distinction between client and server is that the server is typically implemented as a continuous process (daemon); whereas the client is usually implemented as a library. In other words, the client side of an application consists of a call to a routine that executes (sending the request over the network and receiving the result) and then returns and goes on with whatever else it was doing; whereas the server side of an application is a dedicated process that runs continuously—waiting for a request, executing it and returning the answer, then waiting for the next request, and so on. Figure 7 on page 7 illustrates this distinction.

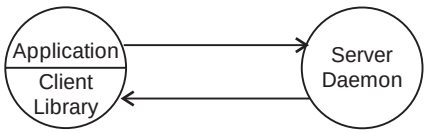

Figure 7. Client as a Library; Server as a Continuous Process

DCE is based on the client/server model. The DCE services are themselves examples of distributed programs with a client and server side. The basic communications mechanism used in DCE, remote procedure call, assumes the presence of a client and a server. Since DCE applications are built using remote procedure call, they are also based on the client/server model of distributed computation.

The Remote Procedure Call Model

One way of implementing communications between the client and server sides of a distributed application is to use the procedure call model. In this model, the client makes what looks like a procedure call. The procedure call is translated into network communications by the underlying RPC mechanism. The server receives a request and executes the procedure, returning the results to the client. One of the DCE technology components, DCE RPC, is an implementation of this model. It is used by most of the other DCE technology components for their network communications. (See “DCE Remote Procedure Call” on page 29 of this manual for more information on remote procedure calls and DCE RPC.)

The Data Sharing Model

Some of the DCE services are based on the data sharing model, in which data is shared by distributing it throughout the system. Like RPC, data sharing assumes the existence of clients and servers. Data sharing, however, focuses on distributed data rather than distributed execution. In RPC, the client’s procedure is executed on the server. In data sharing, the server’s data is sent to the client. For example, if a client wants to access a file, a copy of the file is sent from the server to the client. The client then proceeds to access the file locally. Data sharing can be built on top of RPC, using RPC as the communications mechanism between the client and server, and as the means of transferring data.

Data sharing usually entails having multiple copies of the same data; for example, a master copy of a file on a file server, and a copy of the file on one or more client machines. As a result, copies of data may diverge; that is, a client may make changes to its copy that make the client’s copy inconsistent with the copy on the server. Therefore, distributed services based on the data sharing model usually include mechanisms for keeping copies of data consistent.

In addition, services that implement data sharing must be able to synchronize multiple access to data. For example, two clients may each want to modify a given record in a database. The server that manages the database must either prevent them from making conflicting modifications or decide which modification takes precedence.

Two DCE services are based on the data sharing model. The first is the directory service. Both DCE directory services, CDS and GDS, maintain caches on the client.

The caches contain copies of data that users on the client have recently accessed. Subsequent access to the data can be made locally to the cache, rather than over the network to the server.

The DCE Distributed File Service (DFS) is also based on the data sharing model. A DFS client maintains a cache of files that have recently been accessed by a user on the system. DFS servers distribute and revoke tokens, which represent a client's capability to perform operations on files. Through careful token management, the DFS server can ensure that its clients do not perform conflicting operations on shared files, and that they do not see inconsistent copies of the same file.

Data sharing, like RPC, enables users and programmers to communicate transparently in a distributed system.

The Distributed Object Model

DCE allows for a distributed object model in conjunction with the other DCE models to give a flexible way to distribute functionality and data for client/server applications. In addition, a distributed object model *combines* appropriate functionality with data, by way of distributed objects, in a way that also hides how parts of the distributed application communicates.

Objects are used to model the behavior of all sorts of application entities. In object-oriented terminology, an object is simply an instance of its class. Each object contains member functions (methods) that are only specified in the class as operations. In the distributed object model, a DCE interface is a public set of operations, but the methods of implementation are separate and application specific. (Data types are usually application specific but the interface can specify them as well.) A DCE interface specifies what is known as an abstract base class because the class has a public interface and a hidden implementation.

Object-oriented applications make it easy to hide data and implementation details by using hierarchies of classes and other object-oriented features. Thus object-oriented applications can help minimize the exposure of network details and the special DCE mechanisms of distributed computing. In DCE, the IDL compiler generates a class hierarchy for applications. This hierarchy contains an interface class derived from a DCE RPC base class. The interface class becomes part of an application in such a way that the network details, mechanisms of data transfer, and object location are hidden (encapsulated) in the base class.

DCE interfaces have code generated in C++ to help implement a distributed object model. This means that developers can write object-oriented applications in a more natural way using C++ directly without relying on inadequate or cumbersome C-to-C++ wrapper routines for DCE interfaces.

It has already been stated that the terms client and server are relative roles that an application plays, and not absolute conditions of any particular part of a distributed application. This is also true in the distributed object model.

Architectural Overview of DCE

IBM's Distributed Computing Environment is a layer between the operating system and network on the one hand, and the distributed application on the other. DCE provides the services that allow a distributed application to interact with a collection of possibly heterogeneous computers, operating systems, and networks

as if they were a single system. Figure 8 shows DCE in relation to operating systems, network communications software, and applications software.

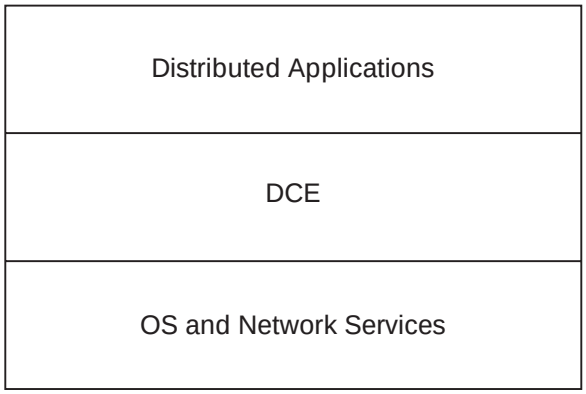

Figure 8. Layering of DCE and Related Software

Several technology components work together to implement the DCE layer. Many of these components provide in a distributed environment what an operating system provides in a centralized (single-node) environment.

Figure 9 on page 10 shows the DCE architecture and its technology components, along with their relationship to applications, underlying system support, and placeholders for future technologies.

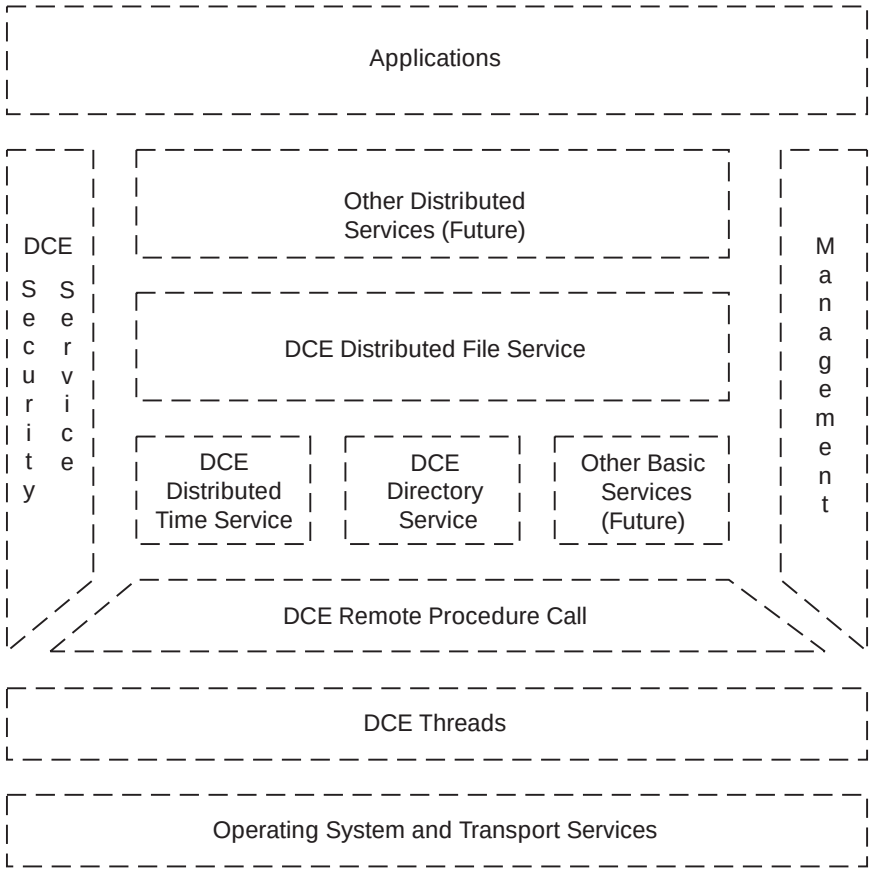

Figure 9. DCE Architecture

Overview of DCE Technology Components

This section gives a short description of each of the DCE technology components. A more in-depth description of each of these components is given in "Chapter 3. DCE Technology Components" on page 25.

DCE Threads supports the creation, management, and synchronization of multiple threads of control within a single process. This component is conceptually a part of the operating system layer, the layer below DCE. DCE threads are used by other DCE components and are also available for applications to use. Currently IBM DCE threads support POSIX 1003.a, draft 4.

The DCE Remote Procedure Call facility consists of both a development tool and a runtime service. The development tool consists of a language (and its compiler) that supports the development of distributed applications following the client/server model. It automatically generates code that transforms procedure calls into network messages. The runtime service implements the network protocols by which the client and server sides of an application communicate. DCE RPC also includes software for generating unique identifiers, which are useful in identifying service interfaces and other resources.

The DCE Directory Service is a central repository for information about resources in the distributed system. Typical resources are users, machines, and RPC-based

services. The information consists of the name of the resource and its associated attributes. Typical attributes could include a user's home directory, or the location of an RPC-based server.

The DCE Directory Service comprises several parts: the Cell Directory Service (CDS), the Global Directory Agent (GDA), and a directory service programming interface. CDS manages a database of information about the resources in a group of machines called a DCE cell. (Cells are described in the next section.) The DCE Directory Service offers the capability to work with Global Directory Services (GDS) provided by other vendors, but an IBM version of GDS is no longer packaged within this product. CDS is accessed using a single directory service application programming interface, the X/Open Directory Service (XDS) Advanced Programming Interface (API).

The DCE Distributed Time Service (DTS) provides synchronized time on the computers participating in a Distributed Computing Environment. DTS synchronizes a DCE host's time with Coordinated Universal Time (UTC), an international time standard.

The DCE Security Service provides secure communications and controlled access to resources in the distributed system. There are four aspects to DCE security: authentication, secure communications, authorization, and auditing. These aspects are implemented by several services and facilities that together constitute the DCE Security Service, including the registry service, the authentication service, the privilege service, the access control list (ACL) facility, the login facility, and the audit service.

The identity of a DCE user or service is verified, or authenticated, by the authentication service. Communications are protected by the integration of DCE RPC with the security service so that communication over the network can be checked for tampering or encrypted for privacy. Access to resources is controlled by comparing the credentials conferred to a user by the privilege service with the rights to the resource, which are specified in the resource's ACL. The login facility initializes a user's security environment, and the registry service manages the information (such as user accounts) in the DCE security database. Security-relevant events can be monitored through the audit service. *Code points* can be set in DCE servers to record events that are deemed to be important to the integrity of the system. For example, the login facility uses the audit service to record logins by DCE users and services.

The DCE Distributed File Service allows users to access and share files stored on a file server anywhere on the network, without having to know the physical location of the file. Files are part of a single, global namespace, so no matter where in the network a user is, the file can be found by using the same name. DFS achieves high performance, particularly through caching of file system data, so that many users can access files that are located on a given file server without prohibitive amounts of network traffic and resulting delays.

The following is AIX specific.

DCE DFS includes a physical file system, the DCE Local File System (LFS), which supports special features that are useful in a distributed environment. They include the ability to replicate data; log file system data, enabling quick recovery after a crash; simplify administration by dividing the file system into easily managed units called *filesets*; and associate ACLs with files and directories.

DCE/File-Access allows users of personal computers running in a Novell NetWare network environment to read and write directories and files on a DCE DFS file server. Users and DCE/File-Access software have DCE identities so DFS file usage is subject to the DCE Security Service's authentication and authorization controls.

The Management block shown in Figure 9 on page 10 is actually not a single component but a cross section of the other components. Each DCE service contains an administrative component so it can be managed over the network. In addition, some of the DCE services themselves provide for management of the distributed system as a whole. For example, users are registered in the security service, and servers' network addresses are registered in the directory service.

The DCE Cell

A DCE cell is a collection of machines, users, and resources managed as a group. For example, imagine an organization made up of several departments, each in a different building and operating on its own budget. Each department in such an organization could have its own DCE cell.

A cell has its own security service, CDS, and optionally, DFS; these services are available cell-wide. The security service for a cell manages the cell's registry, where user account information is kept. Each cell has its own namespace; the cell's CDS manages that namespace and its hierarchy. If DFS is present in the cell, DFS allows remote access to files from anywhere in the cell. Each cell can also have its own DTS, which keeps the clocks on all of the machines in the cell synchronized.

A cell provides a single security domain. Users log into accounts in a cell. ACLs identify users and groups in the cell (they can also refer to users and groups in other cells). A cell also provides a single naming domain. Each cell has a name, and all objects in the cell share that name.

DCE cells can be connected so that they can communicate with each other. Going back to the example, if the different departments' cells are connected, then a user in one department's cell may be able to access resources in another department's cell, although this access would typically be less frequent and more restricted than access to resources within the user's own cell.

Cells connect to each other by means of a global directory service. A cell's name is registered in a global directory service, and the cell is then able to contact other cells registered in that global service. Note that communication between DCE cells is not automatic. Cells that wish to communicate with each other must first establish a trust relationship between their cells' security services; this process is called cross-cell authentication and is described in more detail in "Chapter 2. DCE Configuration" on page 17.

A cell can have more than one name. In this case, one of the cell's names is designated its primary name while the other names are the cell's *alias* names. The cell's primary name is the default name for the cell; that is, it is the name that DCE services return. Cell name aliasing permits a cell to be registered in more than one global namespace. It also provides a way to change a cell's name if the need arises; for example, to respond to organizational changes within the company. For more information on how to create cell name aliases for a cell, see the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Introduction* and the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Core Components*.

A DCE cell can be configured in many ways, depending on its users' requirements. A cell consists of a network connecting three kinds of nodes: DCE user or client machines, DCE administrator machines, and DCE server machines. DCE user machines are general-purpose DCE machines. They contain software that enables them to act as clients to all of the DCE services. DCE administrator machines contain software that enables a DCE administrator to manage DCE system services remotely.

The DCE server machines are equipped with special software enabling them to provide one or more of the DCE services. Every cell must have at least one each of the following servers in order to function:

1. Cell directory server
2. Security server

Note: It is highly recommended that a Distributed Time Server also be included.

Other DCE servers may be present in a given DCE cell to provide additional functionality. A GDA may be present to enable the cell's directory server to communicate with other cells' directory servers; a global directory server may be present to provide X.500 directory service; and distributed file servers may be present to provide storage of files and the special functions of DCE LFS. (See "Chapter 2. DCE Configuration" on page 17 for more detailed information on DCE cell configuration.)

Integration of the DCE Technology Components

One of the benefits of OSF's DCE is its coherence. Although the components themselves are modular with well-defined interfaces, they are also well integrated; the various DCE components each make use of the services of the other components wherever possible. For example, the RPC facility uses the directory service to advertise and look up RPC-based servers and their characteristics, it uses the security service to ensure message integrity and privacy, and it uses DCE Threads to handle concurrent execution of multiple RPCs. DFS uses threads, RPC, the directory service, DTS, and the security service in providing its file service.

In general, the DCE components shown higher in the DCE architecture (see Figure 9 on page 10) make use of the components shown lower in the architecture. For example, DCE Threads is used by most other DCE components, but it does not itself use other components. This ordering is not strictly hierarchical; often two services each depend on the other. For example, the directory service uses the security service, which in turn uses the directory service. The interdependence of DCE components is explained in more detail in "Chapter 4. Integration of DCE Technology Components" on page 77.

Relationship of DCE to Network and System Services

As shown in Figure 8 on page 9, DCE is layered on top of local operating system and networking software. DCE makes certain assumptions about the services provided by the underlying network and operating systems. DCE's requirements for these services are described in the following subsections.

Network Services

In general, DCE is layered over a transport level service, such as User Datagram Protocol (UDP), Transmission Control Protocol (TCP), or ISO TP0-TP4 transport protocols, which is accessed through a transport interface, such as sockets or X/Open Transport Interface (XTI). DCE assumes that all nodes participating in the DCE environment are physically connected by a highly available network. The network can be a Local Area Network (LAN), a Wide Area Network (WAN), or a combination of both.

The DCE architecture supports different types of network protocol families. For example, DCE could be ported to run over Open Systems Interconnection (OSI) protocols. (The IBM DCE reference implementation runs over the Internet Protocol (IP) family.) However, in order for DCE systems to communicate with one another, they must have at least one set of network protocols in common. For example, DCE is not designed to enable a node running only IP protocols to communicate with a node running only OSI protocols.

Finally, DCE assumes the ability to identify a node with a unique network address, and the ability to identify a process with a network endpoint address (for example, a port or T-selector).

Operating System Services

DCE assumes that certain services are available through the underlying operating system, namely the following:

1. Multitasking
2. Timers
3. Local interprocess communications
4. Basic file system operations (VFS layer)
5. Memory management
6. Local security mechanisms (if appropriate)
7. Threads (or the ability to use DCE Threads)
8. General system utility functions

DCE Reference Implementation Dependencies

The previous two subsections listed assumptions made by the DCE architecture. The IBM DCE reference implementation contains additional dependencies on the operating system and network, which are specific to the implementation. These include the use of IP and socket networking services, and UNIX operating system facilities.

DCE Internationalization

DCE internationalization has several aspects:

1. Ensuring character and code set interoperability
2. Preserving character data integrity
3. Ensuring that user-visible messages are easily localized

The next sections discuss these aspects in greater detail.

Ensuring Character and Code Set Interoperability

A character set is a group of characters, such as the English alphabet, Japanese Kanji, or the European character set. To enable world-wide connectivity, DCE guarantees that a minimum group of characters is supported in the DCE. The DCE RPC communications protocol ensures this guarantee by requiring that all DCE RPC clients and servers support the DCE Portable Character Set (PCS). The set of DCE PCS characters consists of the following:

1. 0 1 2 3 4 5 6 7 8 9
2. : ; < = > ? @ [ ] ^ _ ′ ’ ~ { | } ! ″ # $ % & ( ) * + – . / <space>
3. a b c d e f g h i j k l m n o p q r s t u v w x y z
4. A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A *code set* is a mapping of the members of a character set to specific numeric code values. Examples of code sets include ASCII, EBCDIC, JIS X0208 (Japanese Kanji), and ISO 8859-1 (also known as Latin-1.) The DCE RPC communications protocol automatically converts DCE PCS characters between the ASCII and EBCDIC code sets, if necessary. DCE RPC also provides constructs and routines for character and code set interoperability between non-PCS, or international characters. These features permit programmers to write DCE RPC applications that guarantee character and code set interoperability between clients and servers in a DCE that supports a variety of languages and encodings for those languages.

Preserving Character Data Integrity

The DCE components preserve character data integrity because they do not use, or ″mask off,’’ the eighth bit of a character for any purpose. In addition, DCE RPC does not modify user data in any way when it is passed as idl_byte context, unless the application is using the DCE RPC features for automatic code set conversion of non-PCS characters.

Ensuring Easy Localization of User-Visible Messages

The DCE components isolate all user-visible messages into separate message catalogs. Separating DCE messages into distinct message catalogues makes it easier for DCE licensees to localize DCE messages so that DCE users and administrators can view DCE error messages and prompts in their native language.The DCE Messaging API can also be used by DCE application developers to separate and localize their own application messages.

Chapter 2. DCE Configuration

"Chapter 1. Overview of DCE" on page 1 gave some examples of organizations that could benefit from a distributed computing environment. The examples showed that OSF DCE could be useful to organizations for widely varying reasons. Similarly, one organization using DCE could require a DCE configuration that is quite different from the DCE configuration that another organization develops.

This chapter gives an overview of DCE configuration. It describes the basic DCE software configuration components and how they are organized on different types of DCE machines. It then describes some typical DCE cell configurations.

The DCE configuration description in this chapter is based on technical configuration considerations. The packaging of DCE software by OSF and other vendors will involve somewhat different configurations since the packaging is influenced by additional considerations.

Introduction to DCE Configuration

A distributed computing environment consists of machines that communicate over a network and run DCE software. The machines in a DCE environment serve different functions and can therefore run different configurations of DCE software. There are two basic types of machines in a DCE environment:

1. DCE user machine

   Contains DCE software that enables the machine to participate as a client in the DCE environment. A typical example is a user's workstation.

2. DCE server machine

   Runs software that implements one or more of the DCE services. There can be different kinds of DCE server machines. Some examples are a DCE file server machine and a DCE security server machine.

Note: If the machines to be used are of different speeds, it is advisable to configure the primary CDS server on a machine that is as fast or faster than the clients. If both a primary CDS server and a secondary CDS server are used, the primary CDS server should be as fast as or faster than the secondary CDS server and all other CDS clients.

The administration portion of DCE runs on both the user and the server machines. The client admininistration portion runs on the user machine and the server administration portion runs on the server machine.

The different types of DCE machines run different parts of the DCE software. The basic software necessary for any machine to participate in a DCE environment is the DCE user software. Some of the DCE software implements a particular DCE service and is intended to run only on a machine acting as that particular server. For example, the DCE security server software only runs on a machine designated as a DCE security server machine. There are different kinds of DCE server machines. They run their server-specific software, plus the DCE user software.

Figure 10 summarizes the DCE software that runs on different kinds of DCE machines.

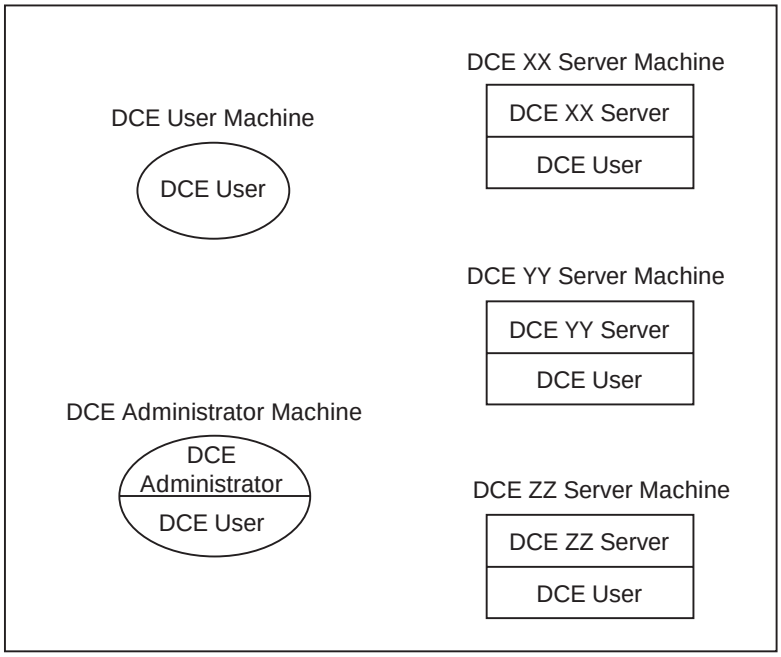

Figure 10. DCE Machines and Software

The following sections describe the DCE software configuration components, machine configuration, and cell configuration in more detail.

Basic Configuration Components

DCE software can be divided into several *configuration components*; that is, parts of the DCE software that are installed in various combinations on DCE machines. Different configuration components are installed on different machines in a DCE environment, depending on what the machine's intended use is. For example, a user's workstation that acts mainly as a client in the DCE environment requires a different set of DCE software from a machine that acts as a DFS file server.

The following description is a model for dividing DCE services into configuration components. The way a service's implementation maps to this model varies from service to service.

First, each DCE service can be divided into two general categories of functionality: user and administration. The user functionality is the service provided to its users; for example, reading a file or searching a database. The *administration* functionality allows administrators to manage the server; for example, stopping and starting server programs or backing up data.

Since the DCE services are based on the client/server model, both the user and administration functions are divided into two parts: the client and server sides. In total, each DCE technology component can be conceptually divided into four configuration components:

1. User client

2. User server
3. Administration client
4. Administration server

As shown in Figure 11, the user client communicates over the network with the user server, and the administration client communicates over the network with the administration server.

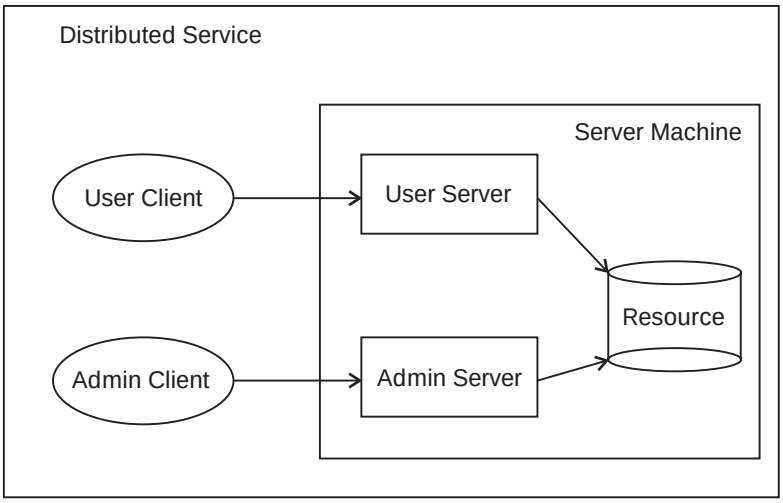

Figure 11. Distributed Service Configuration Components

The user client component is typically installed on DCE users' workstations. The administration client might run only on the workstation used by the administrator of the service. Both the user server and the administration server run on the server machine, since they require access to the resource (such as a database) that the server manages. The user server and administration server may actually run in the same process or be implemented by several processes.

As an example, consider the DCE Security Service. One part of the security service software is the login facility, which sets up a user's security environment. This is an example of a user client. It communicates over the network with the privilege server, which runs on the security server machine. The privilege server is an example of a user server. An example of an administration client in the security service is the sub-commands which relate security registry in dcecp program. The sub-commands include: account, group, organization, principal, registry and user. All of them manage the data in the security database. Each sub-command communicates over the network with the registry server, which runs on the security server machine. The registry server is an example of an administration server.

The software for each of the DCE services, namely the directory service, DTS, the security service, and DFS, can all be divided roughly into these four configuration components.

Because DCE Threads and DCE RPC help to implement the communications between machines, they must be present on every DCE machine whether the machine acts as a client or a server. For this reason, they are always installed and automatically configured when another component (such as Directory client) is selected.

"DCE Machine Configuration Examples" describes how machines participating in a DCE environment are configured by using various combinations of configuration components. "DCE Cell Configuration Examples" on page 21 describes how DCE cells are configured by using various combinations of DCE machines.

DCE Machine Configuration Examples

DCE machine configurations fall into three general categories: user machines, administrator machines, and server machines.

DCE User Machine Configuration

An example of a DCE user machine is a user's workstation. This machine acts as a client to any of the DCE servers, but it does not act as a server itself (with one possible exception noted in the next paragraph). A DCE user machine contains DCE Threads and DCE RPC software so it can communicate with other machines in the DCE environment. In addition, it contains the user client configuration components of all the DCE services (see Figure 12). Part of this software may be present in the form of libraries linked with DCE application software.

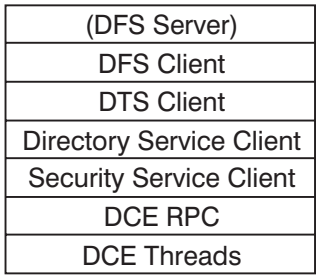

Figure 12. DCE User Machine Configuration

A DCE user machine may also contain DFS server software, although this is not required. This enables the machine not only to access remote files through its DFS client software, but also to export its own file system to other machines through its DFS server software.

We call the software configuration of a typical DCE user machine the DCE user software. In summary, the DCE user contains the following:

1. DCE Threads and DCE RPC
2. User client configuration components of each DCE service
3. DFS server software (optional)

DCE Server Machine Configuration

Some machines in the DCE environment contain special-purpose server software. These are called DCE server machines.

A DCE server machine is configured with the user server and administration server components of a DCE service. It also contains the DCE user software, since a server machine can act as a client to other servers. For example, a DTS server machine contains the DCE user plus the DTS user server and DTS administration

server configuration components. It is not necessary to run one server per machine; two or more types of servers can run on a single machine. Figure 13 shows the configuration of a DTS server machine and the configuration of a second machine acting as both a CDS server and a security server.

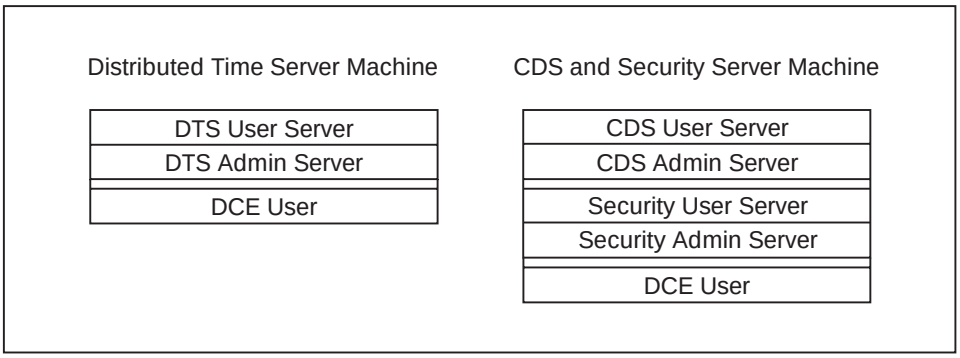

Figure 13. DCE Server Machine Configuration Examples

From now on, we will use the term *server* to mean both the user server and administration server software combined; for example, the term security server means the security user server and the security administration server together.

DCE Cell Configuration Examples

DCE cells are composed of various combinations of DCE machines connected by a network. In order for DCE applications and the DCE services themselves to run, there must be at least one each of the cell directory and security servers in every DCE cell. It is highly recommended that a DTS server also be included in the cell. In addition, a DCE cell can contain any combination of the remaining DCE servers (GDA and DFS) depending on the needs of the DCE users.

The following subsections describe these typical DCE cell configurations:

1. Simple DCE cell
2. DCE cell with DFS file server machine
3. Connected DCE cell

A Simple DCE Cell

Figure 14 on page 22 shows an example of a simple DCE cell. The cell contains seven nodes, each of them running the DCE user software. Four of the nodes are typical workstations; they are running only the DCE user software. One is an administrator's workstation The other two nodes are DCE server machines. One of the server machines is running a security server. The other server machine is running both a cell directory server and a distributed time server. This configuration is a complete, basic DCE cell.

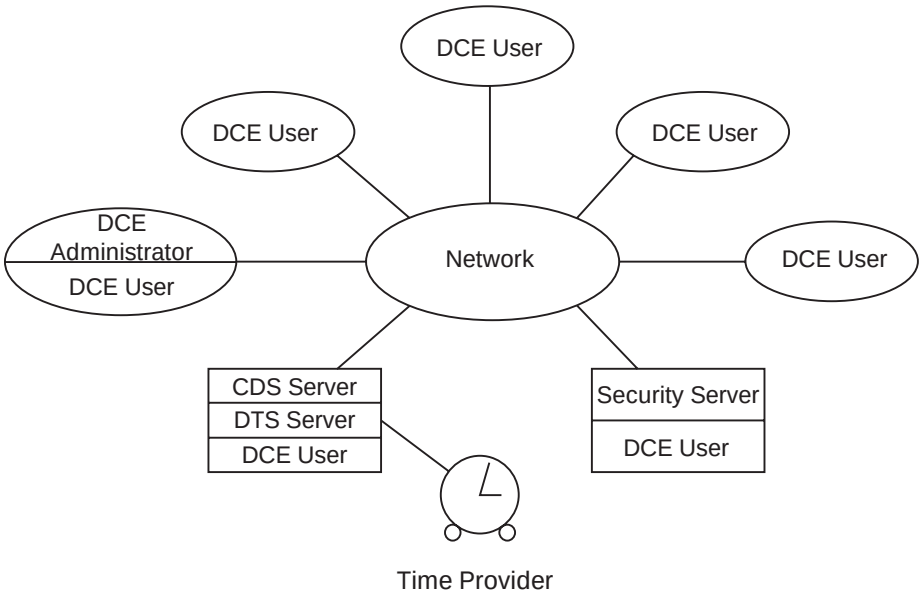

Figure 14. Simple DCE Cell Configuration

Figure 15 shows the same simple DCE cell, this time with a DCE application running in it. Node C is offering the Bank service, and Nodes A and B have the client code for accessing the Bank service. The Bank server has registered itself with CDS so the Bank clients are able to locate it.

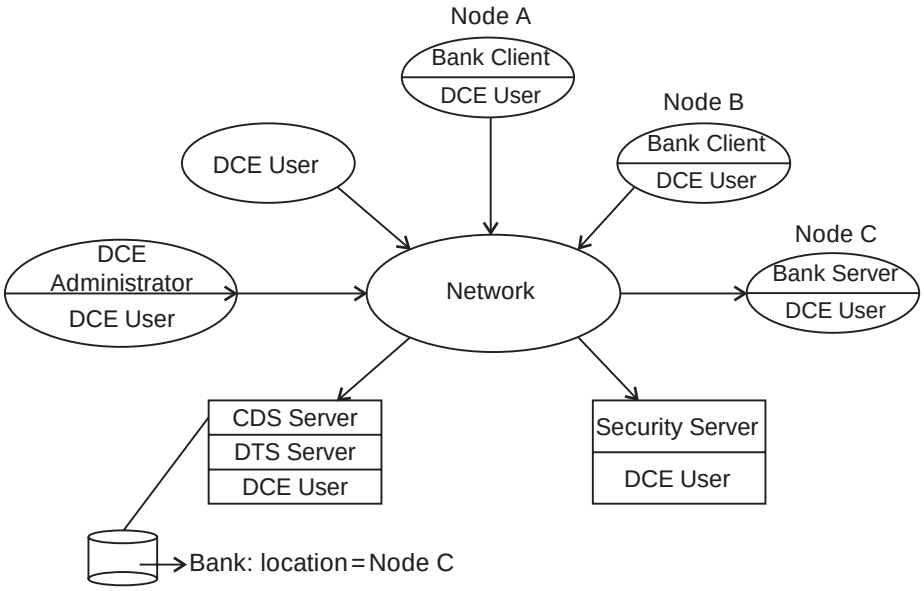

Figure 15. DCE Application in Simple Cell

A DCE Cell with DFS

In order to have full DFS support, including DCE LFS, a DCE cell can contain one or more DFS file server machines (see Figure 16 on page 23). As mentioned in Section 2.3.1, the DCE user is equipped to act as a DFS client and can also export the client's local file system to other machines on the network, using the DFS server software. The DFS file server machine, however, is specially equipped with

DCE LFS, a physical file system that supports distributed file system features such as file replication, online backup, and other advanced administrative support.

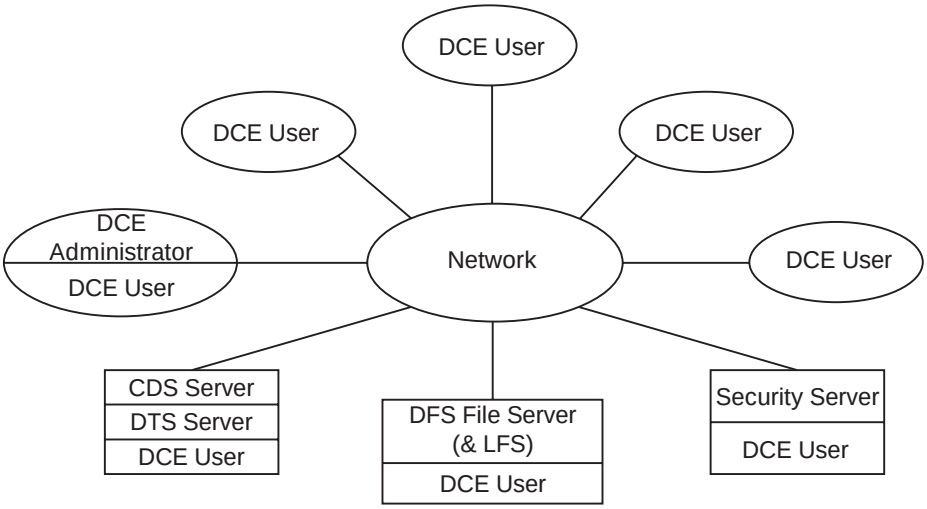

Figure 16. Simple Cell Plus Distributed File Server

A Connected DCE Cell

An organization may wish its DCE cell to communicate with other DCE cells, or with systems outside of DCE. One way to accomplish this is through one of the global directory services that DCE supports:

1. DCE GDS, which is an implementation of the X.500 directory service standard provided as a DCE component

   Note: GDS is not provided in this product. However, you can use GDS with this release if you obtain it from another vendor or if you use the version of GDS provided in DCE 1.3 for AIX.

2. DNS, which is a global directory service that DCE supports but does not provide as a DCE component

A DCE cell is connected to a global directory service when its name is registered in the global directory service's namespace. The cell then establishes a trust relationship between its authentication service and the authentication services of cells that it wants to contact (this step is not necessary for contacting systems without DCE security); this process is called cross-cell authentication. The trust relationship established through cross-cell authentication gives DCE users (and other principals) in the trusted foreign cell authenticated access to resources in the local cell, and vice-versa.

A cell's CDS communicates with CDS servers in foreign cells with the help of an intermediary, GDA. When a GDA machine is added to a DCE cell, the machines in the cell will be able to contact DCE cells and other systems by using X.500 or DNS. Figure 17 on page 24 shows the simple DCE cell with a GDA added to it.

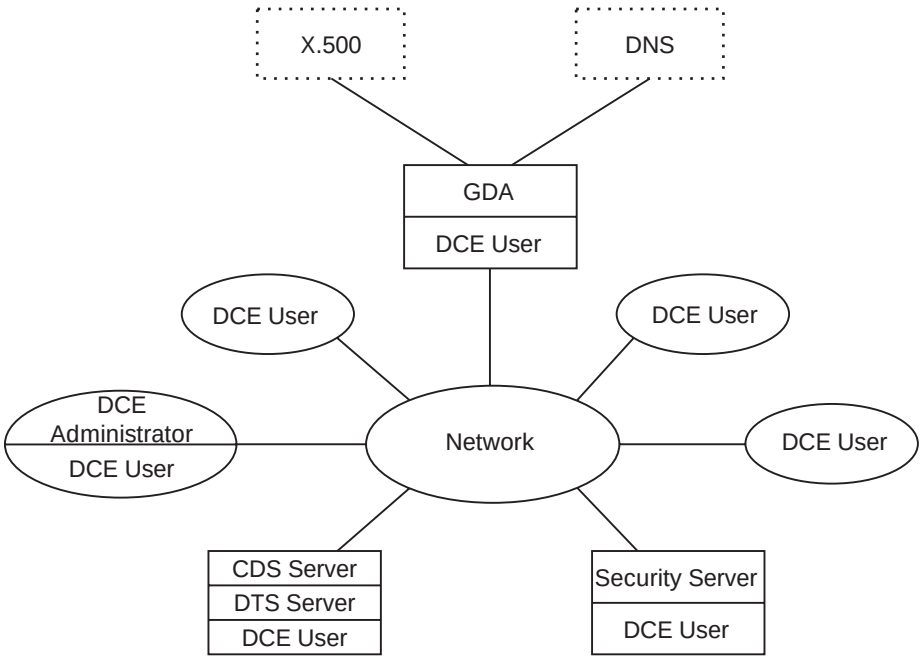

Figure 17. Cell Connected via Global Directory Agent

If a cell contains a global directory server, not only can it access the X.500 namespace through the GDA, but it can also own and administer a portion of that namespace in GDS. For more information on CDS, see “Chapter 3. DCE Technology Components” on page 25 of this manual.

Chapter 3. DCE Technology Components

OSF DCE comprises several technology components:

1. "DCE Threads" on page 26
2. "DCE Remote Procedure Call" on page 29
3. "DCE Directory Service" on page 38
4. "DCE Distributed Time Service" on page 48
5. "DCE Security Service" on page 52
6. "DCE Distributed File Service" on page 59

The DCE components fall into two general categories: distributed programming facilities and *distributed services*. The DCE Threads and RPC components are distributed programming facilities, which include libraries that implement APIs and program development tools.

The remaining DCE components are distributed services. These components consist in part of a daemon, or server process, that runs continuously on a machine and responds to requests sent over the network. They are equipped with administrative subcomponents to manage the service. They also have APIs through which a programmer can access the server.

In general, application programmers deal mostly with the distributed programming facilities: DCE Threads and RPC. Although the distributed services also have APIs for accessing them, the programmer often uses distributed services only indirectly through the RPC facility, which in turn uses the distributed services' APIs. System administrators, on the other hand, deal mostly with the distributed services since they have significant management requirements.

There are also facilities that do not fall under a specific component, but perform services common to multiple components. Among these facilities are the following:

1. Host services
2. Application message service
3. Serviceability
4. Backing store databases
5. DCE control program

This chapter contains sections devoted to each of the technology components. Each of these sections starts with an overview of its technology, along with a description of the pieces that constitute the technology. The sections then describe their technologies from the perspective of different types of users such as the end user's viewpoint, how the programmer develops an application with the technology, and how the administrator manages the technology. Finally, the sections each explain how their technology works, and they describe important benefits or features of the particular technology.

"DCE Cross-Component Facilities" on page 66 gives an overview of the cross-technology facilities. "The DCE Control Program" on page 67 describes the DCE control program (`dcecp`).

The last section of this chapter, “Two DCE Application Examples” on page 68, gives an example of a very simple distributed application, describing the process for developing, installing, and running it.

DCE Threads

In a traditional computer program, there is only one thread of control. Execution of the program proceeds sequentially, and, at any given time, there is only one point in the program that is currently executing. It is sometimes useful, however, to write a program that contains multiple threads of control. For example, some programs lend themselves to being structured as multiple flows of control, some programs show better performance when they are multithreaded, and multiple threads can be mapped to multiple processors when they are available.

A distributed computing environment based on the client/server model and remote procedure call can make good use of the capability for multiple threads of control. For example, when a client makes an RPC call, it blocks until a response is returned from the server. If there are multiple threads of control in the client, then work can continue in another thread while the thread waiting for the RPC response is blocked. On the server side, this same situation applies since a server may itself issue an RPC. In addition, servers often handle the requests of multiple clients. It is sometimes easier to write a well-structured program when each request can be handled by a separate thread of control. Often servers manage information, requiring input/output operations to a storage device. While one server thread is waiting for its input or output operation to finish, another server thread can continue working, improving overall performance.

Using multiple threads puts new requirements on programmers: they must manage the threads, synchronize threads’ access to global resources, and make choices about thread scheduling and priorities. A threads implementation must provide facilities for programmers to perform these tasks.

Threads can be provided by a programming language, an operating system kernel, or a user-space library. DCE Threads is provided as a user-space library; this has implications for its interaction with other software on the system, such as an operating system that delivers signals to or blocks a whole process, rather than just a thread, and preexisting library calls that were not originally written for a multithreaded environment.

The following subsections give an overview of the DCE Threads technology component. They describe the different kinds of functions provided by the technology and how DCE Threads is seen from the end user’s, programmer’s, and administrator’s perspective, focusing particularly on programming with DCE Threads since the application programmer is the main consumer of this technology.

What is DCE Threads?

DCE Threads is a user-level (nonkernel) threads library based on the pthreads interface specified by POSIX in the 1003.4a standard (Draft 4). It consists of an API that gives programmers the ability to create and manipulate threads, as described in “DCE Threads”. The other technology components of OSF DCE assume the availability of threads support. DCE Threads is provided for use on operating systems that do not provide threads already; if a threads package is already available, then DCE Threads may not be needed. DCE Threads can be used as

is—as a user-level threading facility—or it can be mapped to an existing threads facility provided by the host operating system.

End User's Perspective

An end user is not aware whether or not threads are being used in an application, except for a possible difference in performance. An application written with threads may run faster than a single-threaded version of the same application.

Programming with DCE Threads

The distributed application programmer can use threads to help structure a program. However, having multiple threads of control can introduce a higher level of complexity than programming with a single thread of control. Threads must be managed, scheduled, and allowed to communicate with one another in a controlled manner.

Threads Management

In a traditional process, there is only one thread of control, and it is started and terminated implicitly. However, when it is possible to have more than one thread of control, the threads must be created and destroyed explicitly. DCE Threads provides the facilities for doing this.

Threads Scheduling

In the traditional process model, no scheduling is needed since there is only one thread of control, and, whenever the process runs, that thread runs. However, with multiple threads, if there are fewer available processors than the number of threads to be run, some decision must be made as to which thread runs first. This is analogous to the scheduling of processes by the operating system on a timesharing system, except that the threads scheduling is visible to and controllable by the application programmer. (Note that POSIX specifies that scheduling is optional, so systems using their own threads implementations may not include the functionality provided by DCE Threads that is described in this section.)

DCE Threads scheduling is built on two basic, interacting mechanisms:

1. Scheduling priorities
2. Scheduling policies

Each thread has a scheduling priority associated with it. Threads with a higher priority have precedence over threads with a lower priority when scheduling decisions are made. The exact treatment of threads of different priorities depends on the scheduling policy under which they are running.

DCE Threads offers three scheduling policies, the first two of which can be set only by a user with root authority:

1. First-In, First-Out (FIFO)

   The thread in the highest priority category that has been waiting the longest to run is scheduled first. It runs until it blocks, then again the thread that has been waiting the longest runs, and so on. Threads in the highest priority level are run in this first-in, first-out manner, then the threads in the next highest priority level are run FIFO, and so on. You must have root authority to use this scheduling policy.
2. Round-Robin (RR)

All of the threads at the highest priority level are given turns running by timeslicing. That is, one thread runs for a period of time, then it is interrupted and another thread runs for a period of time, and so on, until all threads have had a chance. The process is repeated until all threads in that priority are finished or blocked. Then the threads in the next highest priority level are also run RR until they are all finished or blocked, and so on. You must have root authority to use this scheduling policy.

3. Default

   Each thread is given turns running by timeslicing. Higher priority threads are given longer periods of time to run, but even the lowest priority thread eventually has a chance to run. This is in contrast to FIFO and RR scheduling, in which it is possible for higher priority threads to prevent lower priority threads from running at all.

Thread Communication and Synchronization

Threads communicate through shared variables; that is, one thread sets a variable that another thread later reads. However, if multiple threads are accessing the same variable, incorrect results can occur due to scheduling of threads and race conditions. To resolve this problem, access to shared variables must be synchronized. DCE Threads provides three facilities for synchronizing threads within a process:

1. Mutual exclusion objects (mutexes)
2. Condition variables
3. The `join` routine

The mutex object is used to synchronize access to a given resource, such as a shared variable, by multiple threads. Mutexes ensure that only one thread accesses the resource associated with the mutex at a time, thus the name *mutual exclusion* or mutex.

The mutex works as follows. One mutex object is associated with each shared resource; for example, a shared variable. Before reading or writing the variable, a thread attempts to *lock* the variable's mutex. If it succeeds in locking the mutex, the thread proceeds to access the variable, and then it *unlocks* the mutex.

If a second thread tries to access the object while the first thread is accessing it (the condition that can cause indeterminate results if the shared variable is not protected), the second thread is blocked when it tries to lock the mutex. When the first thread finishes with the variable and unlocks the mutex, the second thread is unblocked and gains the lock for the mutex. It can then proceed to access the shared variable.

The mutex is a facility by which threads can ensure that their access to shared resources is synchronized. The threads may or may not be communicating through the shared data. The second method of thread synchronization, the *condition variable*, is used for explicit communications among threads. This is done through the use of a shared resource—the condition variable—and as a result requires the use of a mutex.

For example, using a condition variable, Thread A can wait for Thread B to accomplish some task. To do this, Thread A *waits* on the condition variable until Thread B signals the condition variable, indicating that the particular task has been accomplished.

Note that, although the condition variable is used for explicit communications among threads, the communications are anonymous. For example, Thread B does not necessarily know that Thread A is waiting on the condition variable that Thread B signals, and Thread A does not know that it was Thread B that woke it up from its wait on the condition variable.

There is another synchronization method that is not anonymous—the `join` routine. This allows a thread to wait for another, specific thread to complete its execution. When the second thread has finished, the first thread unblocks and continues its execution. Unlike mutexes and condition variables, the `join` routine is not associated with any particular shared data.

DCE Threads Exceptions

DCE Threads provides two ways to obtain information about the results of a threads call. One way, specified by the POSIX P1003.4a (pthreads) draft standard, is that status values are returned to the thread. DCE Threads also gives the programmer an alternative to status values. This is provided by the exception-returning interface, which is an extension to the basic POSIX functionality. Exceptions enable routines to ignore status returns when other parts of the program are handling errors.

DCE Threads Administration

There are no administrative tasks associated with the DCE Threads component.

Additional Information on DCE Threads

For additional information on DCE Threads, see the following:

1. The DCE Threads chapters of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide*
2. The DCE Threads, (`3thr`), reference pages of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Reference*
3. The POSIX P1003.4a / Draft 4 *Threads Extension for Portable Operation Systems* Specification
4. The Implementation-Specific Addendum to the POSIX P1003.4a / Draft 4 Specification

DCE Remote Procedure Call

A distributed application based on the client / server model consists of two parts: the client side of the application, which runs on one machine and makes a request for service on behalf of a user, and the server side of the application, which runs on another machine on the network and fulfills the service request. The two pieces of code on two different machines need to be able to communicate across the network. One model for implementing communications between the client and server of an application is the RPC facility.

RPC gives programmers the ability to express an interaction between the client and server of a distributed application as if it were a procedure call; the programmer defines a calling interface and a procedure that implements it, makes a call to the procedure along with any arguments, and receives a return value through the argument list or as the procedure result.

In RPC, as in a traditional local procedure call, control is passed from one code segment to another, and then returns to the original segment. However, in a local procedure call, the code segments are in the same address space on the same machine; whereas, in a remote procedure call, the called procedure runs in a different address space, usually on a different machine than the calling procedure. As a result, arguments and results are passed differently for local and remote procedure calls. In local procedure calls, arguments and return values can be passed on the process's stack. In remote procedure calls, arguments and return values must be packed up into messages and sent to the peer machine over the network. The underlying RPC mechanism makes this look like a procedure call to the programmer.

An RPC facility shields the application programmer from the details of network communications between client and server nodes, such as the following:

1. Fragmentation and reassembly of messages
2. Handling different data formats (such as byte ordering) between different machines
3. Using a directory service to find message recipients
4. Using security services to ensure secure communications

Programmers using RPC do not need to rewrite applications in order to port them to different architectures, operating systems, communications protocols, or languages. RPC provides a high-level programming model to the distributed application programmer, hiding communications details, and removing nonportable system and hardware dependencies.

The following subsections give an overview of the DCE RPC technology component. They describe the components that constitute the technology, and how DCE RPC is seen from the end user's, programmer's, and administrator's perspectives, focusing primarily on programming with RPC since the application programmer is the main consumer of this technology. The subsections also describe the steps involved in the execution of a remote procedure call. They describe the ways in which DCE RPC frees the programmer from system software and hardware dependencies, and then list additional sources of information on DCE RPC.

What is DCE RPC?

DCE RPC is a facility for calling a procedure on a remote machine as if it were a local procedure call. To the application programmer, a remote call looks (almost) like a local call, but there are several RPC components that work together to implement this facility, including the Interface Definition Language (IDL) and its compiler, a Universal Unique Identifier (UUID) generator, and the RPC runtime, which supports two RPC protocol implementations. One RPC protocol operates over connection-oriented transports such as the Transmission Control Protocol/Internet Protocol (TCP/IP), and the other RPC protocol operates over connectionless transports such as the User Datagram Protocol/Internet Protocol (UDP/IP).

An end user does not see RPC at all, and the minimal amount of administration involved in RPC can usually be handled by the server-side application code, such as advertising an application server in the DCE Directory Service. It is the application programmer who most comes into contact with the RPC component. Since many of the DCE components are themselves client/server applications, they use RPC as their basis for distributed communications.

The components that constitute the DCE RPC are as follows:

1. IDL and the IDL compiler

   An RPC interface is described in DCE IDL. The IDL file is compiled into object code via the IDL compiler. The object code is in two main parts: one for the client side of the application, and one for the server side.

2. The RPC runtime library

   This library consists of a set of routines, linked with both the client and server sides of an application, which implement the communications between them. This involves the client finding the server in the distributed system, getting messages back and forth, managing any state that exists between requests, and processing any errors that occur.

3. Authenticated RPC

   DCE RPC is integrated with the DCE Security Service component to provide secure communications. Levels of security can be controlled by the RPC application programmer through the authenticated RPC API. (See "Programming with DCE Security" on page 55 for more information on authenticated RPC.)

4. Name Service Independent (NSI) API

   DCE RPC is integrated with the DCE Directory Service component to facilitate the location of RPC-based servers by their clients. The NSI routines allow a programmer to control the association, or binding, of a client to a server during RPC.

5. DCE host daemon

   The `dced` program runs on every DCE machine. It includes (among other things) an RPC-specific name server called the endpoint mapper service, which manages a database that maps RPC servers to the transport endpoints (in IP, the ports) that the server is listening for requests on.

6. DCE control program

   `dcecp` is a tool for administering DCE.

7. UUID facilities

   These are ancillary commands and routines for generating UUIDs, which uniquely identify an RPC interface or any other resource. The `uuidgen` program can optionally generate an IDL template for a service interface, along with a unique identifier for the interface.

End User's Perspective

The end user does not come in direct contact with DCE RPC, but does see the end result in the form of

1. The availability of distributed applications built using RPC
2. The ability to use remote resources accessed via RPC

An end user who is browsing through the namespace may also notice the names of RPC-based servers, since these servers advertise themselves to their clients through the DCE Directory Service.

Programming with DCE RPC

This section provides a brief overview of the process a programmer goes through in using DCE RPC to write an application. For an example of how this process applies to a simple application, see "Two DCE Application Examples" on page 68

of this manual. For a more detailed description of the DCE RPC programming process, see the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide.*

Figure 18 shows an overview of the DCE RPC application development process. The dashed boxes indicate application code written by the DCE programmer. The other boxes indicate compiled code or code generated automatically for the programmer by DCE RPC.

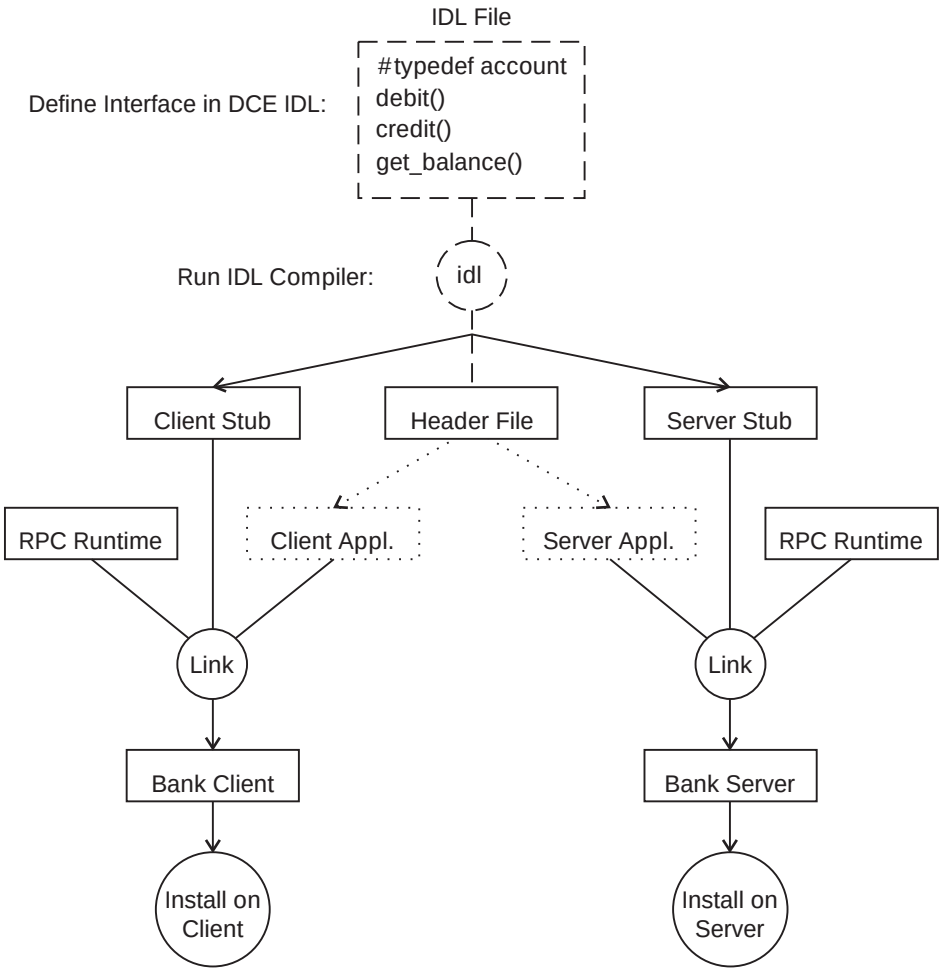

Figure 18. DCE RPC Programming Process

The IDL File

First, the application programmer defines the RPC interface, and associated data types, using IDL. An *interface* is a group of operations that a server can perform. This grouping is similar to a module or library in a conventional programming language; that is, a group of operations defined in a single file or unit. For example, a Bank service might perform operations to debit, credit, or read the balance of an account. Each of those operations and their parameters must be defined in the IDL file. The collection of Bank service operations—debit, credit, read balance—together form the Bank service interface.

The process of defining RPC operations is similar to defining the input and output of a local procedure call, except that in IDL only the calling interface is defined, not the implementation of the procedure. (An IDL interface definition is similar to an ANSI C prototype definition.)

Next, the programmer compiles the IDL file with the IDL compiler. The compiler produces output in a conventional programming language, which is the C language in the DCE offering and then calls the appropriate compiler to produce object code. The output of the compilation consists of a *client stub*, a *server stub*, and a header file. The client and server stubs are routines that make the remoteness of the operation transparent to the caller or callee of the operation.

The Client Side

For the client side of the application, the programmer writes application code that makes calls to the operations in the IDL file. The client stub code is linked with this application code and (along with the RPC runtime code) performs the tasks that turn what looks like a procedure call into network communications with the server side of the application. Usually the client side of the application contains a relatively small amount of RPC code.

The Server Side

For the server side, the programmer writes application routines that implement the operations defined in the IDL file. For example, in the banking application, a database lookup might implement the operation to read an account balance. The server stub, generated by the IDL compiler, is linked with the server application code. The server stub unpacks the arguments and makes the call to the application routine as if the client program had called it directly. The server side of the application contains the bulk of the RPC code that needs to be written by the distributed application programmer.

Binding

In order for the client to send an RPC to the server, it must be able to find the server. This process is called *binding*. A client may have some direct way of knowing what server it needs to communicate with; for example, it may get this information from a file, a value hardcoded into its program, an environment variable, or a user. A more flexible way for a client to find a server is to take advantage of DCE RPC's use of the DCE Directory Service.

A client can find a server by asking the directory service for the location of a server that handles the interface that the client is interested in (in our example, a Bank server). In order for the directory service to be able to give the client this information, a server must first advertise itself in the directory service. The server adds itself to the namespace, along with information about what interfaces it implements, what protocols it uses to communicate with, and where it is located. This way, a server can move, or there can be multiple servers implementing a given interface, without affecting the client. The client can still go to the directory service, a well-known central source of information, and find out where the server is located.

The DCE programmer does not make calls directly to CDS; binding is supported by the NSI API, an RPC-specific name service layer. Calls to this library are made by the client side of an application in order to look up binding information for a server based on various criteria, such as the type of service, the objects it manages,

and the interfaces it supports. The server side of an application calls this library to advertise information about itself to the namespace where clients can find it.

The Endpoint Mapper Service of the DCE Host Daemon

There are two parts to a server's location: the network address of the machine it resides on and the transport-specific address of the process—the network endpoint (for example, a UNIX port). The machine location is fairly stable, so its address can reasonably be entered into CDS. The network endpoint, however, can change each time the server process is started. Instead of making frequent changes to CDS to update a server's endpoint address, DCE RPC uses a specialized type of directory service, the endpoint mapper service, a service of dced. When a server starts, it registers its endpoint(s) with dced. Most servers will register an endpoint for each transport protocol supported on the host (for example, TCP and UDP).

Every machine that runs an RPC server also runs dced. The dced process always uses the same network endpoint, so its process address is well known. Therefore, once a client knows what machine a server is running on, it can find the endpoint mapper running on that same machine. It can then ask the endpoint mapper for the network endpoint of the server process. This process is shown in Figure 19 (read the messages, shown in quotes, in clockwise order).

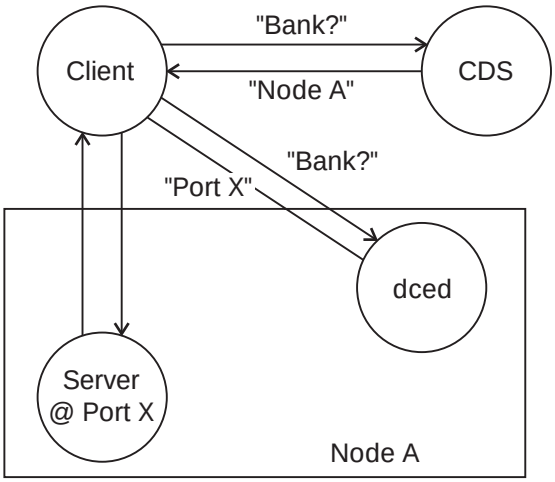

Figure 19. Client Finds Server via CDS and dced

DCE RPC Administration

A few administrative tasks must be performed when running a distributed application using RPC. The application server executes most of these tasks when it first starts. As described in the previous section, the server registers its (dynamically assigned) listening endpoint with dced. The server also advertises information about itself and the interfaces it supports in the DCE Directory Service.

Nonautomated RPC administration is minimal. The administrator must ensure that each DCE machine has a DCE host daemon running on it. An administrator may be involved in registering servers in the namespace, but this can also be done from server code upon initialization as just described. Usually, an administrator will be needed to change the ACL on the directory where the server will register so that

the server has write permission. The DCE control program, dcecp, allows an administrator to (among many things) control the dced and administer RPC information in the namespace.

An administrator may be involved in installing a new RPC-based application. In particular, the server side of the application must be started before it can begin receiving and servicing requests. The administrator may arrange for the server process to be started each time the machine is booted, for example.

How an RPC Call Works

A short ''walk-through'' of what happens during an RPC call may help clarify the RPC concept and how it works. This section describes the RPC call shown in Figure 20. (This description is somewhat simplified. The use of dced is not shown.)

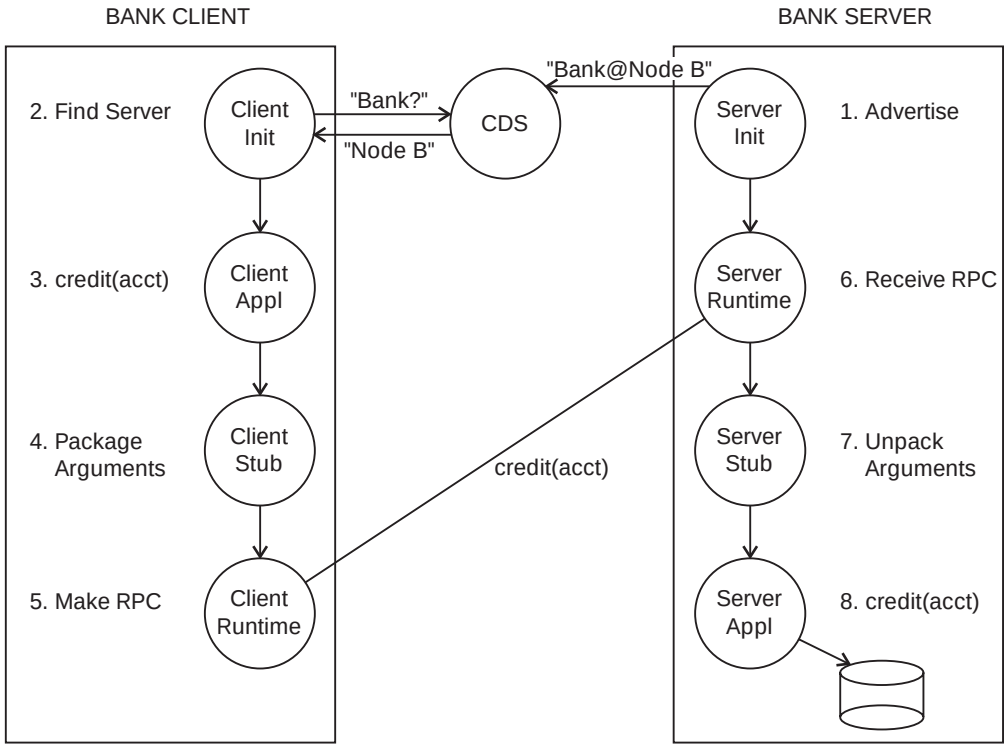

Figure 20. RPC Runtime Process

On the server side, the Bank server process is started up. Before it begins its continuous cycle of receiving and servicing requests, the server process advertises its location in CDS (see Step 1 in Figure 20). In this way, when a client queries the directory service for a bank server, it will be able to find it. After initialization, the server listens for a request to come in from a client over the network. This call to wait for client requests is a call to the RPC runtime, since the runtime handles network communications.

Eventually, a user on the Bank client machine invokes the bank application. The Bank client initialization code calls the RPC runtime to find a server offering the Bank service (see Point 2). The Bank client application code makes a call to a remote procedure; for example, a call to a procedure that credits a bank account (3). This results in a call to the client stub code. The stub transforms the arguments

of the call into a network message (4). It then calls the client's RPC runtime library, which sends the message to the server (5).

Back on the server side, the RPC request is received by the RPC runtime, which has been waiting for a client request (6). The runtime passes control, and the received packet, to the server stub. The stub unpacks the arguments sent by the client (7) and passes them to the appropriate operation by making a procedure call to it. At this point, the server application code that implements the requested operation is called. The operation is performed; that is, the account is credited (8).

The RPC reply (not shown in the figure) returns in the reverse direction. The Bank server application procedure returns the results of the credit operation to the stub. The stub packs up the return parameters and passes the resulting message to the RPC runtime to send off to the client over the network. The server then waits for the next client request to come in.

The client's runtime receives the server's reply. The client stub then unpacks the received network message, arranging returned parameters such that, when the client application call to RPC returns, it looks like it has just performed a local procedure call.

The mechanisms in both the client and server stubs and the runtime library are transparent to the application programmer. The programmer writes the application calls to the RPC operations on the client side, and provides implementations for those operations on the server side, but the network communications code is generated automatically.

System Independence

There are several ways in which using DCE RPC frees a programmer from dependence on other parts of a system. It provides portability across programming languages, data transfer syntax mechanisms, transport and network protocols, and operating system and architecture platforms.

1. Language independence

   DCE RPC is *language independent* in the sense that the stubs generated by the IDL compiler can be called by programs written in any traditional programming language, provided that the language follows the calling conventions that the stub expects. The DCE IDL compiler generates stubs that use the C language calling conventions. A client written in FORTRAN, for example, can call an IDL stub in the same way that it calls any local C procedure. It can then make a remote call to a server (possibly written in another language) that contains the server stub generated from the same IDL file as the client stub was generated from.

2. Data representation independence

   The default data representation format is the DCE Transfer Syntax, which is currently the Network Data Representation (NDR). It allows various representations for different types of data, including multiple encodings for characters, integers, and floating-point numbers. It is *multicanonical*; that is, there are several canonical formats that can be used. The sender chooses one of these formats (in most cases, it will be the sender's native data representation), with information about what representation it has chosen. The receiver transforms data into its own format, if it is different from the format the data was sent in. This model optimizes for the case when both sender and receiver

use the same data representation, a frequent occurrence. (Note that this data transfer is handled by the RPC software and is not visible to the application programmer.)

The DCE RPC architecture allows the use of transfer syntaxes other than DCE Transfer Syntax (although the only transfer syntax currently provided in the OSF implementation is DCE Transfer Syntax). For example, data could be formatted according to the ISO ASN.1/BER specification and sent over the wire in that way.

3. Protocol independence

   Independence of RPC, transport, and network protocols is achieved as follows. The DCE RPC offering includes two different RPC protocols. The first runs over connection-oriented transport protocols; the second runs over connectionless (datagram) transport protocols. The programmer can specify the underlying RPC protocol, but the semantics of RPC calls are the same whether the RPC is running over a connectionless or connection-oriented transport. Another RPC protocol could be used in place of these two DCE RPC protocols; for example, when ISO defines an RPC standard, it could be used transparently as a third RPC protocol under the DCE RPC interfaces.

   Servers identify themselves to the directory service according to the interface they support and the protocols they use. In this way, a client can look up a server that uses network protocols that are compatible with those that the client supports.

4. Machine independence

   Because DCE RPC uses the DCE transfer syntax, distributed applications are *machine independent*. DCE transfer syntax allows machines to transfer data even when their native data representations are not the same.

5. Operating system independence

   Finally, DCE RPC offers independence from the *local operating system*. The application programmer is not directly using the networking system calls provided by the local operating system. By being one level of abstraction up from this layer, the programmer is insulated from networking system calls that are operating system specific.

Additional Information on DCE RPC

For additional information on DCE RPC, see the following:

1. The RPC chapters of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide* and the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide*
2. The (3rpc) reference pages of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Reference*
3. The (1rpc) and (8rpc) reference pages of the *IBM DCE Version 3.1 for AIX and Solaris: Administration Commands Reference*
4. The RPC-related error messages in the *IBM DCE Version 3.1 for AIX and Solaris: Problem Determination Guide*

DCE Directory Service

A distributed system may contain many users, machines, and other resources, along with large amounts of data, all geographically dispersed. The distributed system's attributes, such as the number of users, location of servers, and contents of data, are continuously changing. It is difficult to keep track of this potentially large, geographically distributed, rapidly changing system.

A directory service can help solve this problem. When a directory service is available, it is no longer necessary to maintain local copies of information, such as databases of users, hosts, and server locations, on each system. Instead, an application queries the directory service when it needs information. In a sense, the directory service is the most basic of all distributed system services since it is used to find the information needed for accessing other services.

The next section gives an overview of the DCE Directory Service architecture. "DCE Cell Directory Service" on page 43 and "DCE Global Directory Agent" on page 46 describe the DCE Directory Service components—CDS and GDA. "The Directory Service Interfaces" on page 47 describes the directory service application programming interface.

DCE Directory Service Architecture

The DCE Directory Service is a distributed, replicated database service. It is distributed because the information that forms the database is stored in different places; for example, information about one group of users and resources might be stored in one directory server, while information about a second group of users and resources is stored in a different directory server. The directory service is replicated because information about a given name or group of names can be stored in more than one location, for higher availability.

Note: If the machines to be used are of different speeds, it is advisable to configure the primary CDS server on a machine that is as fast or faster than the clients. If both a primary CDS server and a secondary CDS server are used, the primary CDS server should be as fast as or faster than the secondary CDS server and all other CDS clients.

The directory service database consists of a hierarchical set of names, the *namespace*, which have associated attributes. Given a name, its associated attributes can be looked up in the directory service. For example, given the name of a print server, the directory service can return the printer's location. The directory service gives distributed system users a well-known, central place to store information, which can then be retrieved from anywhere in the distributed system.

Overview of Directory Service Components

There are two components that together make up the DCE Directory Service:

- DCE Cell Directory Service (CDS)
- DCE Global Directory Agent (GDA)

The X/Open Directory Service (XDS) application programming interface is used to access the directory service components. A brief overview of the directory service components and interface is given in this section; subsequent sections in this chapter describe them in more detail.

DCE Cell Directory Service: CDS stores names and attributes of resources located in a DCE cell. It is optimized for local access since most directory service queries are for information about resources within the same cell as the originator of the query. CDS is replicated; this is important for a local directory service since the directory service must be highly available. There must be at least one CDS server in each DCE cell. Figure 21 shows three organizations, each with its own DCE cell.

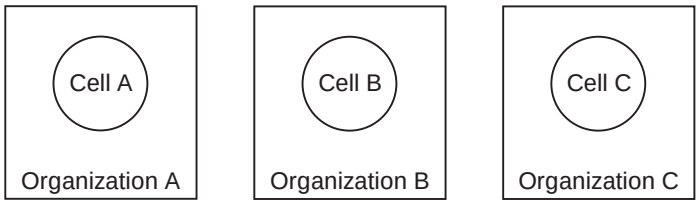

Figure 21. Three One-Celled Organizations

DCE Global Directory Agent: GDA is the intermediary between a cell's CDS and the rest of the world. It takes a name that cannot be found in the local cell and finds the foreign cell in which the name resides, using GDS, DNS, or CDS, depending on where the foreign cell is registered.Figure 22 gives a functional picture, including the use of GDAs.

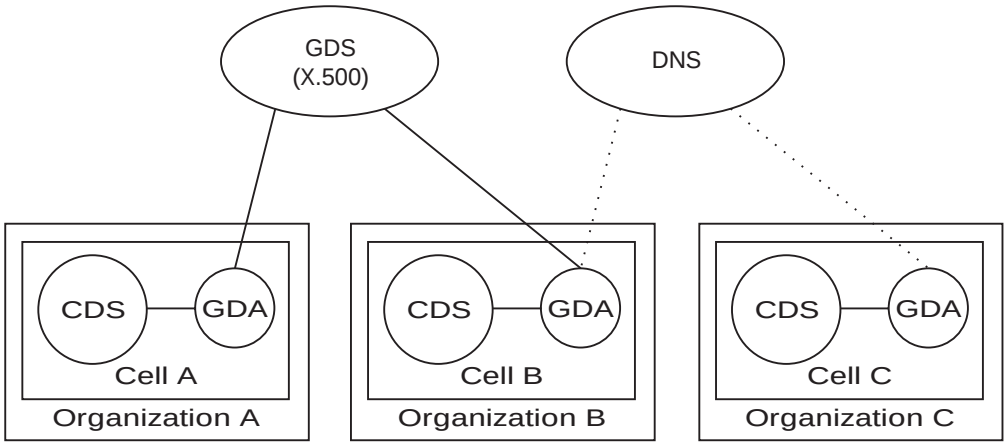

Figure 22. Use of Global Directory Agents

DCE Directory Service Application Programming Interface: DCE programmers use the externalized XDS API to make all directory service calls. The XDS library knows, based on the format of the name to be looked up, whether to direct the calls it receives to GDS or to CDS (see Figure 23 on page 40). In this release, it directs calls only to RPC unless you have GDS from another vendor or are using IBM DCE release 1.0.3. XDS uses the X/Open Object Management (XOM) API to define and manage its information.

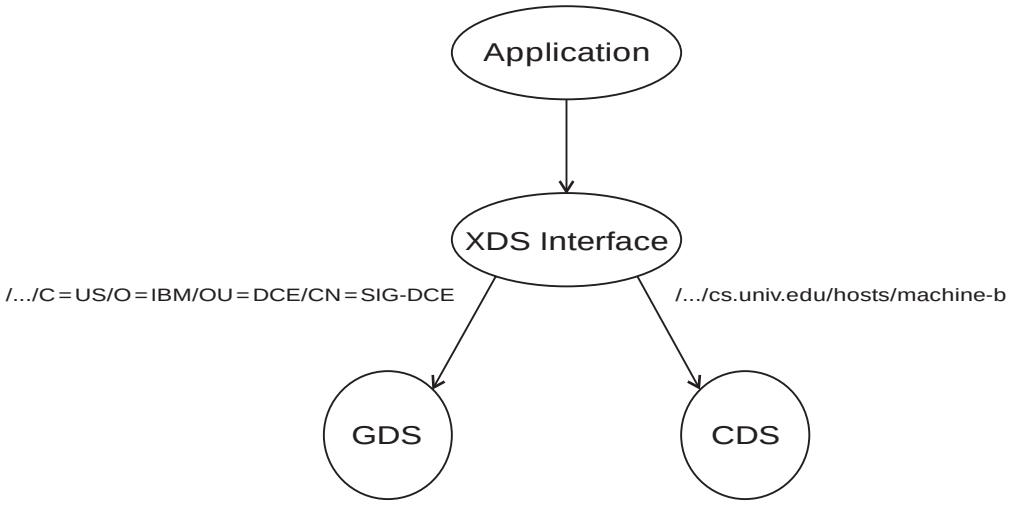

Figure 23. XDS: Interface to GDS and CDS

The DCE Namespace

The DCE namespace is the set of names used by the DCE Directory Service. It is hierarchical, similar to the structure of a UNIX file system. Names can be typed or untyped, reflecting the different name formats supported by the two global directory services: GDS and DNS. GDS names are typed; that is, they consist of a type and a value separated by an = (equal sign). A name such as `/C=US/O=ABCcompany`, names an object that exists in GDS. An untyped name consists only of values such as `abc.com`; DNS names use this format.

Figure 24 shows the root of the DCE namespace, indicated by the `/...` prefix, and four cell entries below it.

The two cells on the left, `/.../C=US/O=IBM/OU=DCE` and `/.../C=CA/O=B-College/OU=EE-Department`, are in the X.500 namespace, while the two cells on the right, `/.../company_b.com` and `/.../cs.univ.edu`, are in the DNS namespace.

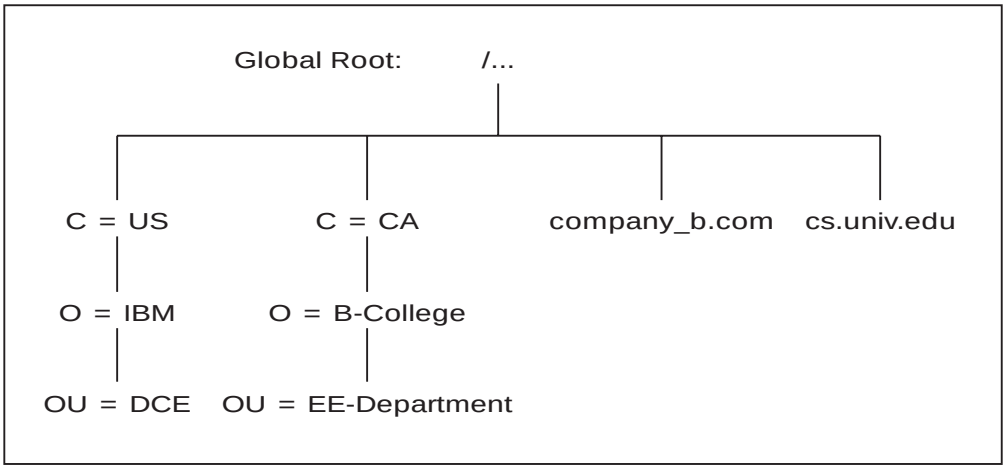

Figure 24. Four Cells in DCE Global Namespace

Figure 25 shows the top of a typical DCE cell namespace. It contains an entry for security information, an entry for the cell's DFS file system, an entry for subsystems such as DCE services, an RPC profile entry, and an entry for host names. (See the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Introduction* for more information on the cell namespace.)

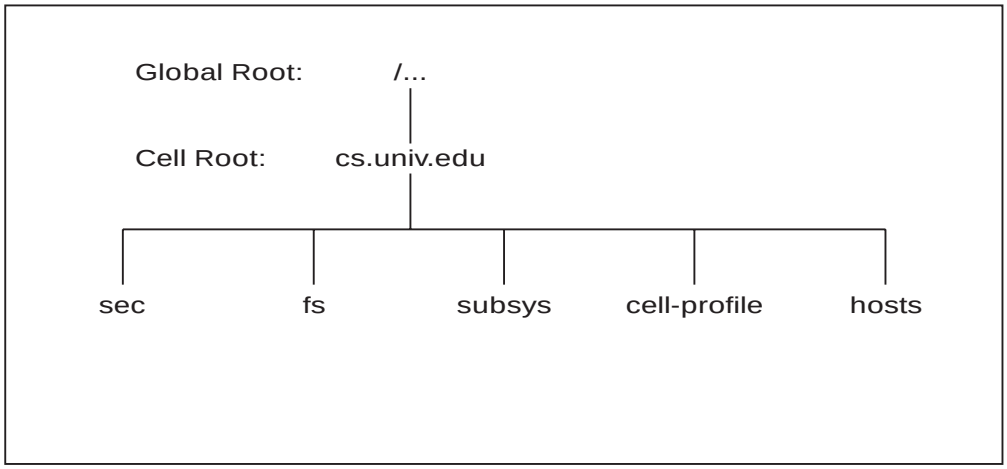

Figure 25. Top of a Typical DCE Cell Namespace

The following is a list of examples of typed and untyped names:

```
/.../C=US/O=IBM/OU=DCE/sec/principals/snowpaws
/.../C=US/O=IBM/OU=DCE/fs/usr/snowpaws
/.../cs.univ.edu/sec/principals/ziggy
/.../cs.univ.edu/fs/usr/ziggy
```

The `/...` prefix indicates that the name is a global name. The first two names are typed names using X.500 syntax, and the second two names are untyped names using DNS syntax. The first name in each set indicates the name of a user in an authentication database; the second name in each set is the user's home directory in a file system.

In each of the name examples, there is a global component and a local component. The global component consists of a cell name, which is registered in a global directory service. In one case, the cell is an entry in the X.500 namespace: `/.../C=US/O=IBM/OU=DCE`. In the other case, the cell is an entry in the DNS namespace `/.../cs.univ.edu`. The remainder of the name is an entry in the cell's namespace; for example, `/fs/usr/snowpaws`.

The names listed here reside in the DCE cell namespace, but it is also possible to maintain names in the X.500 namespace by using GDS. An example of this kind of name is `/.../C=US/O=IBM/OU=DCE/CN=SIG-DCE`. This name could be used, for example, for an electronic mail list.

Viewpoints on the Directory Service

The DCE Directory Service looks very different to the end user, programmer, and administrator. This section takes a brief look at the directory service from each of these three perspectives.

End User's Perspective: The DCE Directory Service is one of the few DCE technologies that is visible to the end user. An end user can use the CDS browser to look through the cell's namespace. A frequent use of the namespace is in referencing the file system. The pathname for a file in a foreign cell is partially a pathname in the directory service, as in the example `/.../cs.univ.edu/fs/usr/ziggy` given previously.

Application Programmer's Perspective: DCE application programmers do not necessarily need to interface directly with the directory service, since a frequent use of the directory service—to look up the location of a server—can be done automatically by DCE RPC. Programmers who do use the directory service interact with it through the X/Open directory service interface. XDS provides facilities for adding, deleting, modifying, and looking up names and their attributes.

Programmers use XDS for accessing both DCE directory services—CDS and GDS. However, the programmer needs to understand the different types of names used for different namespaces, and be aware of some XDS calls that are not available when CDS is being used. An example is the `search` query, which is possible in GDS, but not in CDS.

Administrator's Perspective: Two Directory Services and an Intermediary: Unlike the end user and application programmer, the directory service administrator is aware of the cell's directory service configuration, since the two directory services are administered separately. The administrator manages the CDS server, GDA, and the GDS server, if the cell has one.

Related Services: Registration and Lookup Path

There are two services in DCE that are distinct from, but related to, the DCE Directory Service. The first is registration. In naming an object in a distributed system, it is useful to have a unique name for the object. DCE provides a facility for generating UUIDs, which are used to name DCE objects such as RPC interfaces, users, and computing resources. More information on UUIDs is contained in the RPC chapters of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide.*

A second service that is related to directory service is the ability to specify a path through the directory service for looking up names. In DCE, this capability is provided by RPC profiles. Profiles can be used, for example, to look up names relative to a certain location. If a user wants to look up a printer based on the printer's proximity to the user, for example, a profile may specify that a directory service lookup for a printer begin in the local cell, and then, if a printer is not found, look in the two neighboring cells, and so on. For more information on RPC profiles, see the RPC chapters of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide.*

Specialized Naming Services

The DCE namespace is not contained entirely in the DCE Directory Service. Other system services contain parts of the namespace and some of them require their own specialized naming services, which supplement the DCE Directory Service. These include

1. DCE host daemon (`dced`)

   Maintains a database of local data that is essential for operating in a DCE environment, such as the host's cell name. It also keeps a database of server configuration information that it uses to manage DCE server operation on a

host; for example, starting or stopping a DCE server. An example of a name in the DCE host daemon part of the namespace is as follows, where `video_clip` is the name of a DCE application server:

```
/.../cs.univ.edu/hosts/gunther/config/srvrconf/video_clip
```

2. DCE Security Service database

   Keeps a database of DCE principals (users and servers) and information related to them such as their passwords. An example of a name in the security part of the DCE namespace is `/.../cs.univ.edu/sec/principal/ziggy`.

3. DFS Fileset Location server database

   Maintains a database that maps DFS filesets to the DFS file server machines they are kept on. An example of a name in the DFS part of the DCE namespace is `/.../cs.univ.edu/fs/usr/ziggy`.

All three of these databases are accessed by way of junction points in CDS. A junction point contains binding information that enables a client to connect to a server's database outside of CDS. In these three databases, **/.:/hosts**, **/.:/sec**, and **/.:/fs** are the junction points to the DCED database, the Security database, and the DFS database, respecitvely.

DCE Cell Directory Service

One of the two directory services underlying the XDS API is DCE CDS. The following subsections describe CDS in terms of the data elements that it deals with and the components that implement it. They then describe how CDS handles replication, caching, and data consistency. Finally, they describe CDS from the end-user, programmer, and administrator perspectives.

What is CDS?

DCE CDS is made up of several components, including the CDS server, CDS clerk, and CDS administration programs.

1. CDS server

   Runs on a node containing databases of directory information. It responds to queries from clients by accessing the database. Each clearinghouse (see 3 on page 44) has its own database.

2. CDS clerk

   Runs on the client node and serves as an intermediary between client applications and CDS servers. One of the clerk's most important functions is to maintain a cache of information acquired through directory queries. The clerk stores the response to a query in its cache so that the next time a related query is made, the information is already available on the client node, and no network communications with the CDS server are necessary. The cache is written to disk periodically, so it persists even if the clerk process dies or the client node crashes.

3. CDS administration programs

   Carry out CDS administrative tasks. Administrators can use the DCE control program, `dcecp`, for the majority of CDS administrative tasks. There are also two administrative programs included in the CDS technology—the CDS browser and the CDS control program. The CDS browser, which end users as well as CDS administrators can use, is a CDS client application that allows you

to inspect the cell's namespace. The CDS control program, `cdscp`, enables administrators to control CDS servers and data.

Figure 26 shows a client application that sends a request to the CDS clerk, which in turn communicates with the CDS server. The server performs a database lookup or update, depending on the request. The response is then sent back to the client application.

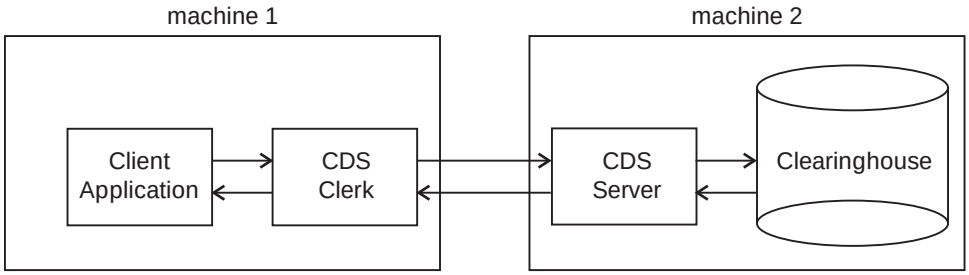

Figure 26. CDS Client and Server Machines

The CDS Database

CDS information is contained in three types of data elements:

1. Directory entries

   A directory entry consists of a name and its attributes. One example is the name of an application server, whose attributes include the interface it exports and its location on the network.

2. Directories

   A CDS *directory* is a logical grouping of CDS information; it is a collection of directory entries. The directory is the administrative unit for replication. There can be one or more copies, or *replicas*, of a given directory. CDS directories are in a hierarchical relationship to one another; each directory has a parent directory and can also have child directories.

3. Clearinghouses

   A clearinghouse is a physical CDS database; it is a collection of directory replicas. The clearinghouse is the database on a CDS server machine that the CDS server accesses in order to respond to its requests.

As an example of how the different types of CDS data elements relate to one another, imagine a directory P, which contains all the information about the printers in a given cell. This directory contains one directory entry per printer. The administrator of the cell may decide that the information contained in the P directory is important enough that it needs to be replicated on more than one CDS server, so, if one server goes down, the P information can still be found on the other server. To that end, replicas of the P directory might be kept in two clearinghouses: one replica in Clearinghouse A, and the other in Clearinghouse B.

Replication and Data Consistency in CDS

A directory service must be highly available since other services depend on it. It must also be fast. CDS achieves these two goals through the replication of directories and caching of directory entries. It also provides mechanisms for keeping various degrees of consistency among copies of data.

There are two types of directory replicas in CDS:

1. Master replica
2. Read-only replica

There is exactly one master replica of a given directory, and any kind of operation can be performed on it. The only operations that can be performed on a read-only replica are those limited to read access to the directory; no updates can be made to this type of directory replica. There can be zero or more read-only replicas.

CDS provides two methods for maintaining data consistency among replicas of a directory:

1. Immediate propagation
2. Skulking

With immediate propagation, a change made to one copy is immediately made to other copies of the same data. Immediate propagation is used when it is important for all copies of a directory to be consistent at all times.

A skulk happens periodically (for example, every 24 hours), and is done on a per-directory basis. All changes made to the given directory are collected and propagated in bulk to all clearinghouses that contain replicas of the directory. If a skulk cannot complete (that is, if one or more of the nodes containing a replica to be updated is down), then an administrator is notified and the skulk is attempted again later.

Caching is also a form of replication and therefore leads to the problem of keeping multiple copies of information consistent (or in this case, dealing with the fact that cached information may be out of date). As mentioned previously, the CDS clerk caches directory information so that information will be available on the local node rather than having to repeatedly query directory servers. When the client application wants to make sure it has the latest information, CDS allows the application to bypass the clerk's cache and go directly to the CDS server for information by allowing the user to set the confidence levels.

End User's Perspective

An end user may be interested in perusing the cell namespace to look for information contained in CDS. This can be done using the CDS browser.

Programming with CDS

Programmers can access CDS through XDS. They also use CDS indirectly when they use the name service routines of the RPC API.

CDS Administration

In general, CDS administrators use `dcecp` to administer CDS; They can also use `cdscp` in the few cases where `dcecp` does not provide the necessary administrative function. Administrators can use the CDS browser to inspect CDS data. CDS administrative tasks include monitoring CDS servers, managing access control on CDS directories, and specifying replication and update of CDS data.

Additional Information on CDS

For additional information on CDS, see the following:

1. The `dcecp` and CDS chapters of the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Core Components*.

2. The (8dce) and (8cds) reference pages of the *IBM DCE Version 3.1 for AIX and Solaris: Administration Commands Reference*.

DCE Global Directory Agent

DCE GDA acts as an intermediary between the local cell's directory service and the global directory services. In particular, the GDA handles CDS calls that are directed to foreign cells. The foreign cells must be registered with one of the two global directory services that DCE supports: the X.500 directory service or the Domain Name Service (DNS).

What is GDA?

DCE GDA is a process that interfaces between CDS and GDS or DNS. Programmers access the GDA indirectly through the XDS API. GDA administration consists simply of starting and stopping the GDA process.

At least one GDA must be present in a DCE cell in order for that cell to acquire directory service information from other DCE cells.

GDA and Other Directory Service Components

Figure 27 shows how the GDA relates to other directory service components.

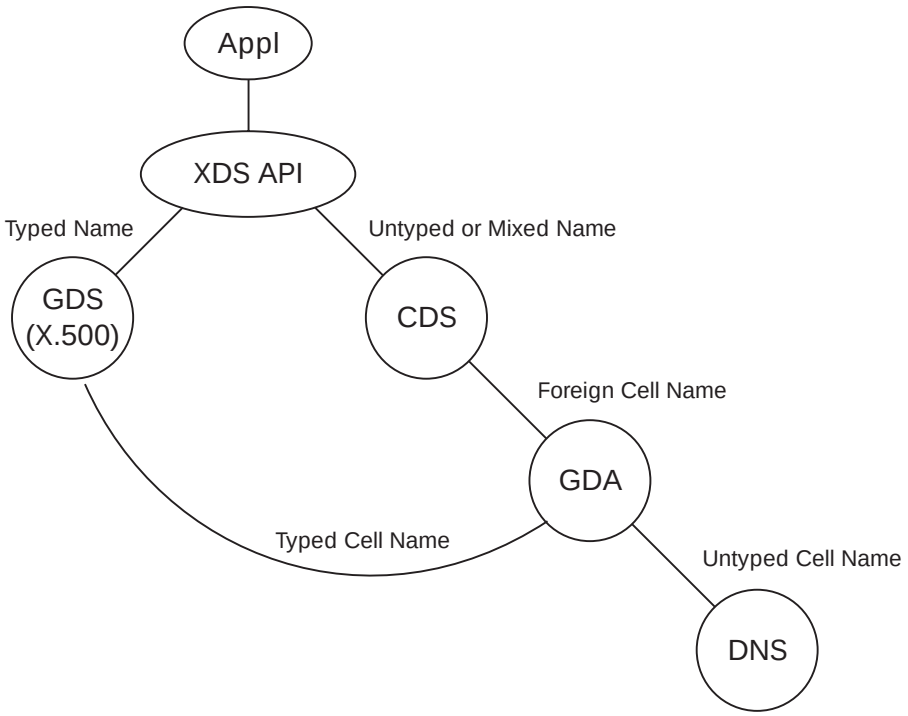

Figure 27. GDA and Other Directory Service Components

The application uses XDS to make a directory service call. If a typed name such as /.../C=US/O=IBM/OU=DCE/CN=SIG-DCE is to be accessed, then the query is passed to GDS. If the name to be accessed is an untyped name, such as /.../cs.univ.edu/fs/usr/ziggy, or a partially typed name, such as /.../C=US/O=IBM/OU=DCE/fs/usr/snowpaws, then the query is passed to CDS. If the

name is a local name, contained in the local CDS, then the query is handled by the local CDS server. If it is a name that resides in a foreign cell, it is passed to the GDA.

The GDA determines whether the foreign cell is registered in X.500 or DNS, based on the format of the name. It then contacts a GDS server or DNS server to find the foreign cell. Once the foreign cell is found, information about that cell is cached so that subsequent lookups of names in that cell do not require the involvement of a global directory server. Finally, the CDS server in the foreign cell is contacted to handle the query about the name.

Additional Information on DCE GDA

For additional information on DCE GDA, see the GDA sections of the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide*.

The Directory Service Interfaces

The XDS and XOM APIs provided by the DCE Directory Service are based on X/Open specifications. APIs are not really separate components (every DCE component includes APIs to access it), but we call them out separately in this case because programmers use the directory service APIs to access both DCE directory services (CDS and GDS).

The XOM Application Programming Interface

XOM is an interface for creating, deleting, and accessing objects containing information. It is an object-oriented architecture in that each object belongs to a particular class, and classes can be derived from other classes, inheriting the characteristics of the original class. The representation of the object is transparent to the programmer; the object can only be manipulated through the XOM interface, not directly. XOM is used to create the objects that make up the directory service.

XOM defines basic data types, such as Boolean, string, object, and so on. It defines an *information architecture*, including concepts such as objects, their attributes, and their classes. It also provides definitions of routines for manipulating objects.

The XDS Interface

The XDS API is used by DCE programmers for accessing information in the DCE Directory Service, whether the information is managed by CDS or GDS. It uses the XOM interface for defining and handling the information objects that compose the directory. These objects are passed as parameters and as return values to the XDS routines. The XDS API contains routines for managing connections with a directory server; reading, comparing, adding, removing, and modifying entries; listing directories; and searching for entries. Some extensions to the X/Open standard that the DCE XDS API provides include provisions for security and cache management.

Additional Information on XDS and XOM

For additional information on the XDS and XOM interfaces, see the following:

1. The XDS and XOM chapters of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide*
2. The (3xds), (4xds), (3xom), and (4xom) reference pages of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Reference*

3. X/Open CAE Draft 1 (May 1991) Specification, API to OSI Object Management (XOM)
4. X/Open CAE Draft 1 (May 1991) Specification, API to Directory Services (XDS)

DCE Distributed Time Service

A distributed computing system has many advantages but also brings with it new problems. One of them is keeping the clocks on different nodes synchronized. In a single system, there is one clock that provides the time of day to all applications. Computer hardware clocks are not completely accurate, but there is always one consistent idea of what time it is for all processes running on the system.

In a distributed system, however, each node has its own clock. Even if it were possible to set all of the clocks in the distributed system to one consistent time at some point, those clocks would drift away from that time at different rates. As a result, the different nodes of a distributed system have different ideas of what time it is. This is a problem, for example, for distributed applications that care about the ordering of events. It is difficult to determine whether Event A on Node X occurred before Event B on Node Y because different nodes have different notions of the current time.

DCE DTS addresses this problem in two ways:

1. DTS provides a way to periodically synchronize the clocks on the different hosts in a distributed system.
2. DTS also provides a way of keeping that synchronized notion of time reasonably close to the *correct* time. (In DTS, correct time is considered to be UTC, an international standard.)

These services together allow cooperating nodes to have the same notion of what time it is, and to also have that time be meaningful in the rest of the world.

Distributed time is inherently more complex than time originating from a single source since clocks cannot be continuously synchronizing, there is always some discrepancy in their ideas of the current time as they drift between synchronizations. In addition, indeterminacy is introduced in the communications necessary for synchronization since clocks synchronize by sending messages about the time back and forth, but that message passing itself takes a certain (unpredictable) amount of time. So in addition to being able to express the time of day, a distributed notion of time must also include an *inaccuracy* factor; that is, how close the timestamp is to the real time. As a result, keeping time in a distributed environment requires not only new synchronization mechanisms, but also a new form of expression of time—one that includes the inaccuracy of the given time. In DTS, distributed time is therefore expressed as a range, or interval, rather than as a single point.

What is DTS?

There are several different components that constitute DCE DTS:

1. Time clerk
2. Time servers
   a. Local time server
   b. Global time server
   c. Courier time server

   d. Backup courier time server
3. DTS API
4. Time-Provider Interface (TPI)
5. Time format, which includes inaccuracy

The active components are the time clerk and different kinds of time servers. There are two interfaces: a programming interface (DTS API) and an interface (TPI) to an external time-provider. Finally, DTS defines a new format for expressing time.

Time Clerk

The time clerk is the client side of DTS. It runs on a client machine, such as a workstation, and keeps the machine's local time synchronized by asking time servers for the correct time and adjusting the local time accordingly.

The time clerk is configured to know the limit of the local system's hardware clock. When enough time has passed that the system's time is above a certain inaccuracy threshold (that is, the clock may have drifted far enough away from the correct time), the time clerk issues a synchronization. It queries various time servers for their opinion of the correct time of day, calculates the probable correct time and its inaccuracy based on the answers it receives, and updates the local system's time.

The update can be gradual or abrupt. If an abrupt update is made, the software register holding the current time is modified to reflect the new time. Usually, however, it is desirable to update the clock gradually and, in this case, the tick increment is modified until the correct time is reached. In other words, if a clock is normally incremented 10 milliseconds at each clock interrupt, and the clock is behind, then the clock register will instead be incremented 11 milliseconds at each clock tick until it catches up.

Figure 28 shows a LAN with two time clerks (C) and three time servers (S). Each of the time clerks queries two of the time servers when synchronizing. The time servers all query each other.

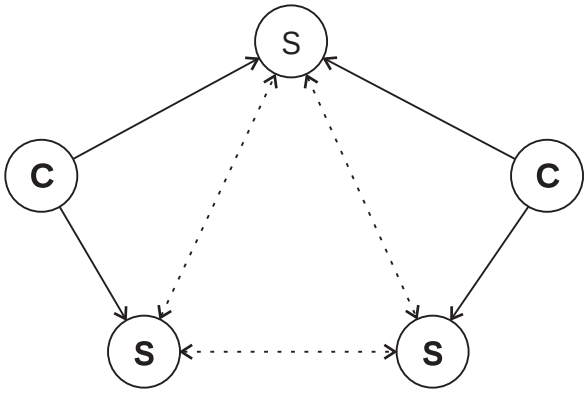

Figure 28. DTS Time Clerks and Servers

Time Servers

A time server is a node that is designated to answer queries about the time. The number of time servers in a DCE cell is configurable; three per LAN is a typical number. time clerks query these time servers for the time, and the time servers query one another, computing the new system time and adjusting their own clocks

as appropriate. One or more of the time servers can be attached to an external time-provider (described later in this section).

A distinction is made between local time servers (time servers on a given LAN) and global time servers. This is because they are located differently by their clients. A client may need to contact a global time server if, for example, the client wants to get time from three servers, but only two servers are available on the LAN. In addition, it may be desirable to configure a DTS system to have two LAN servers and one global time server synchronizing with each other, rather than just having time servers within the LAN synchronizing with each other. This is where couriers are needed.

A courier time server is a time server that synchronizes with a global time server; that is, a time server outside the courier's LAN. It thus imports an outside time to the LAN by synchronizing with the outside time server. Other time servers in the LAN can be designated as backup courier time servers. If the courier is not available, then one of the backup couriers serves in its place.

Figure 29 shows two LANs (LAN A and LAN B) and their time servers (S). In each LAN, one of the time servers acts as a courier time server (Co) by querying a global time server (G) for the current time.

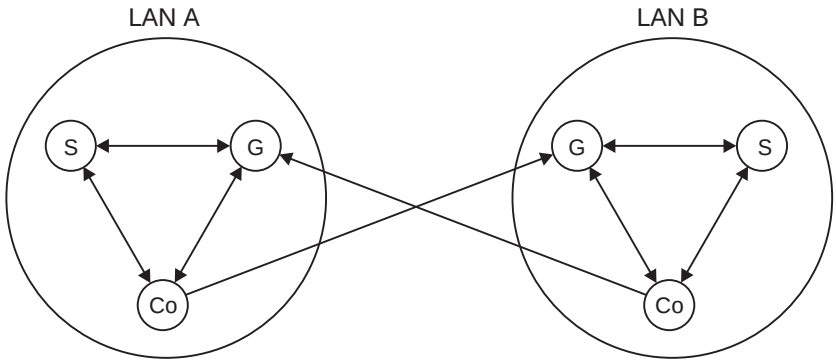

Figure 29. Local, Courier, and Global Time Servers

DTS Application Programming Interface

DTS provides an API library that allows programmers to manipulate timestamps. For example, programmers can obtain a timestamp representing the current time, translate timestamps to different formats, and compare two timestamps.

Time-Provider Interface

So far, all the components described are those supporting the synchronization of a distributed system's clocks. There must also be a way to ensure they are synchronized to the *correct* time. The notion of the correct time must come from an outside source, which is the external time-provider. This may be a hardware device such as one that receives time from radio or telephone sources. This external time is given to a time server, which then communicates it to other servers. Such an external time-provider can be very accurate. If no such device is available, the external time source can be the system administrator, who consults a trustworthy time source and enters it into the system. This cannot, of course, be as accurate as an automatic time source, but it may be sufficient in some cases.

DTS supports the ability to interface with an external time-provider through the time-provider interface. The external time-provider itself, however, is a hardware device (or a person), and is therefore outside the scope of DCE.

DTS Time Format

The time format used in DTS is a standard one: UTC, which notes the time since October 15, 1582, the beginning of the Gregorian calendar. This time is interpreted using the Time Differential Factor (TDF) for use in different time zones. For example, the TDF in New York City is -5 hours. The TDF for Greenwich, England is 0.

End User's Perspective

From a user's point of view, the advantage of having a distributed time service is that more applications work as expected in a distributed environment. For example, the UNIX `make` program compiles new binary files if it discovers that the source file has been changed since the last time the binary was compiled. In a distributed system, this may not work properly if the source is on one machine and the binary is on another, and the two machines have different ideas of what time it is (and of what time it was when their files were updated). Having DTS means that the nodes have roughly the same notion of time, and the `make` program works as expected.

Programming with DTS

There are two ways a programmer can be affected by the presence of DTS in a system. It is possible for an application to retrieve the time from the system in the same way as before DTS was introduced. But with DTS, the system's time is more correct and is synchronized with other nodes' clocks in the distributed system.

It is also possible for the programmer to use the DTS API to directly deal with distributed time. Since DTS time is represented differently than single-node time—it includes inaccuracy—new routines are provided for comparing times and for converting from DTS time format to the native system's time format. The API also includes routines for retrieving the current time, performing calculations on times, and handling time zone information.

If programmers choose to use DTS directly, they must handle a new contingency when comparing times. When asking the question ″Which time is earlier, Time A or Time B?'' it is possible to get the answer ″We do not know.'' When the two time intervals overlap, there is no way of determining which occurred first. Programmers can handle this in two ways: they can ignore the inaccuracy and compare the two median times, or (the safer alternative) they can acknowledge that either time could have been first and take the more conservative action. For example, if it cannot be determined when running the `make` program, whether the source or the executable was modified last, the compilation can be rerun just in case the source was modified after the executable was generated.

DTS Administration

Administering a distributed time service is more involved than administering the time in a single node. In a single node, time administration typically consists of setting the time and date when a system is first started up and then updating the time occasionally to compensate for clock drift.

DTS requires more set-up and configuration work for the administrator, although, once it is up and running, the administrative maintenance tasks are infrequent.

Interaction with the Network Time Protocol

The Network Time Protocol (NTP), an Internet recommended standard that is widely used in the Internet, is another method of synchronizing time in a distributed environment. It is possible for NTP servers to provide time to a DTS system, and for DTS servers to provide time to an NTP system. See the chapter on NTP in the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide*for additional information.

Additional Information on DTS

For additional information on DCE DTS, see the following:

1. The DTS chapters of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide*and the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide*
2. The (3dts) reference pages of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Reference*
3. The (8dts) reference pages of the *IBM DCE Version 3.1 for AIX and Solaris: Administration Commands Reference*

DCE Security Service

A distributed computing environment brings with it new security requirements beyond those found in a nondistributed system. In a nondistributed system, the operating system can be trusted to protect resources from unauthorized access. This is not the case in open distributed systems, however. Communications take place over an accessible network, where messages between machines can be observed or forged. A new security system is required in order to control access to resources in a distributed environment. In DCE, resource protection is provided by the DCE Security Service or, alternatively, the Generic Security Service (GSS).

What is the DCE Security Service?

The DCE Security Service comprises several parts, including the authentication service, the privilege service, the registry service, the ACL facility, the login facility, the password strength service, and the audit service.

1. Authentication service

   This service enables two processes on different machines to be certain of one another's identity, or *authenticated*. On a timesharing system, this functionality is provided in part by the operating system kernel. However, since a local host's operating system cannot necessarily be trusted in a distributed system, an authentication service is necessary in a distributed computing environment.

2. Privilege service

   Once a server has verified the identity of the user who is making a request, it still needs to determine whether the user should be *authorized*, or granted the requested access to a resource that the server controls. This functionality is provided by the DCE authorization service called the privilege service. It forwards in a secure way the information that a server needs to know in order to determine what permissions it should grant to the user.

Both the authentication service and the privilege service are used in conjunction with DCE RPC and the login facility, so the typical application programmer does not interact with them directly, but instead uses authenticated RPC.

3. Registry service

   The registry service is a replicated service that manages the cell's security database. The security database contains entries for security entities, which are called principals. A principal can be a user or a server, for example. The database also contains information associated with each principal; for example, encryption keys, which are used in authentication, authorization, and encryption of messages. The registry service enables administrators to access and modify the database of DCE users.

   The extended registry attribute (ERA) interface allows the registry schema to be modified so that user-defined attributes can be associated with registry objects.

4. ACL facility

   DCE ACLs are lists of users who are authorized to access a given resource. For example, a user can put a colleague on an ACL for a certain file, thereby granting the colleague permission to read and write the file. DCE ACLs are associated with many DCE resources: files, entries in the directory service, and entries in the security service. DCE ACLs are based on the POSIX 1003.6/Draft 3 specification. An ACL API allows programmers to manipulate ACLs, and `dcecp` allows users to modify ACLs associated with resources they own.

5. Login facility

   The DCE login facility initializes a user's DCE security environment. It authenticates the user to the security service by means of the user's password. The security service returns security credentials, which are then used to authenticate the user to distributed services that are accessed during the user's session, such as DFS or other applications. The login facility permits log in using the following authentication protocols:

   a. The public key certificate login protocol, which provides the highest level of security
   b. The third-party protocol, which is less secure than public key protocol
   c. The timestamps protocol, which is less secure than the third-party protocol
   d. The DCE Version 1.0 protocol, which is the least secure protocol. This is provided solely for compatibility with DCE Version 1.0 clients.

6. Password Strength service

   The password strength server checks password-change requests to determine whether the requests can be accepted. When a password-strength server is configured for an account, the password-strength server enforces the password composition rules defined by the parameters used to start this server *instead of* the rules specified in the Settings notebook of the registry or policy group.

7. Audit service

   The audit service detects and records the execution of DCE server operations that are relevant to the maintenance of a secure distributed computing environment. The audit service records the *event* in a log file called an *audit trail file*. DCE application programmers build auditing into their DCE servers by designating security-relevant operations as *code points* for which auditing is required, and using an audit API to establish auditing of those operations. The DCE Security Service and DCE DTS also use the audit service to track and record the use of their security-critical operations. Administrators can use the audit service *event class* and *filter* mechanisms to organize and tailor the recording of events into audit trail files. In addition these audit events can be routed to the DCE Event Management Service (EMS). For more information

about the Event Management Service see the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Core Components* .

How DCE Security Works

This section gives an overview of how the DCE security services and facilities interact to provide a secure distributed computing environment. Figure 30 shows this process. The presentation in this section is a simplified view of the transactions that actually take place.

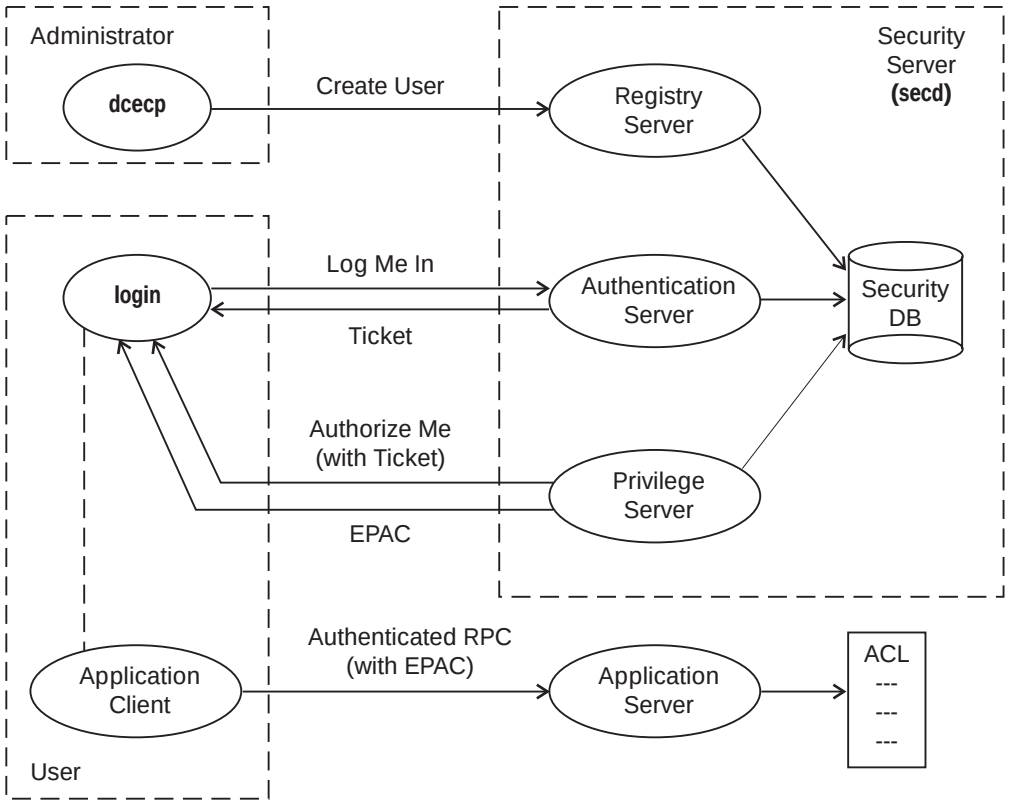

Figure 30. DCE Security Interactions

When a DCE cell is first created, the DCE security administrator runs a program to create an initial DCE security database. The administrator then starts up a DCE security server, called secd, on the same machine as the database. Using dcecp, the administrator creates user accounts in the security database.

After the administrator has created an account for a user, the user can participate in a secure DCE system. Typically a user logs into the account at the beginning of a session. The login facility interacts with both the authentication server and the privilege server. It sends a request for authentication credentials to the authentication server. The authentication server sends back the authentication credentials, called a ticket. The authentication server's reply is encrypted using the user's password; so, if the user can supply the right password, the reply can be decrypted and the ticket can be accessed. Tickets are used by clients to authenticate themselves to servers; that is, to prove that clients are really who they say they are.

Next, the login facility sends the ticket to the privilege server. The privilege server returns authorization credentials, called an extended privilege attribute certificate

(EPAC). The EPAC contains authorization information specific to the user, such as which groups the user belongs to. EPACs are used to authorize users; that is, to help a server decide whether users should be granted access to resources that the server manages. When the login facility has finished running, the user has a security environment and can communicate in a secure way with application servers.

For example, if the user runs an application client, the application client can use authenticated RPC to communicate with the application server. The application server receives the RPC-based request, which includes the user's EPAC. The server inspects the user's authorization credentials and the ACL associated with the resource the user wants to access. If, for example, the ACL says that any user in the group `friends` can access the resource, and the user's EPAC shows that the user is in the `friends` group, then the server will give the user access to the resource.

The authentication and authorization information that is sent over the network is all encrypted so that only the intended recipients are able to decrypt and read the messages. If desired, the application data can be encrypted as well. This prevents any unauthorized user from being able to read data that is sent over the network.

The encryption method used in DCE is secret key encryption, in which two parties share a secret—the encryption key. Using that key, they can encrypt and decrypt each other's messages. (For information on the generation, transfer, and use of encryption keys in DCE security, see the security chapters of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Introduction and Style Guide* and *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Core Components*.

Finally, although it is not shown in Figure 30 on page 54, all of the security service events just discussed (creating a user, logging in, obtaining a ticket, and so on) are recorded and logged in an audit trail file by the audit service daemon, `auditd`. In addition, if the application server has been designed to use the audit service, and the operation that the application client is requesting has been designated an audit code point, then the audit service logs the execution of the server operation on behalf of the requesting client.

End User's Perspective

Much of the DCE security mechanism occurs without the knowledge of users; for example, secure communications take place without the user's intervention. There are several ways, however, in which users do come in contact with DCE security. One instance is when users type in their passwords to authenticate themselves to DCE, usually at login time. Another case is when they change access to resources they own, using `dcecp`. Finally, a user notices the security service in action when he or she is denied unauthorized access to resources.

Programming with DCE Security

Typically, a DCE programmer uses DCE RPC to implement a distributed application. DCE security is integrated with RPC, so in some cases the programmer does not need to access security services directly. However, programming interfaces to the security service are available to the programmer who needs them. They include the ACL, audit, login, extended registry attribute, security credentials,

and registry APIs, along with the API for authenticated RPC. This section gives an overview of programming with authenticated RPC and DCE ACLs.

Authenticated RPC

DCE RPC and DCE security cooperate to provide authentication, authorization, and secure communications. In order to use authenticated RPC, the client must already have a security environment, such as that set up by the login facility. The server side of the application registers its name and the type of authentication service it supports. In DCE, two categories of authentication service are supported: secret key authentication, which is based on Kerberos (see "DCE Security and Kerberos" on page 58), or no authentication.

The client makes a call to indicate which authentication service, protection level, and authorization service it wants to use for RPC communications with a given server. The authentication service can be either secret key authentication or no authentication. The protection level ranges from authentication at the beginning of an RPC session, to authenticating each message or packet, to ensuring that a packet has not been modified in transit, to encrypting all user data. In general, the more secure the protection level, the higher the price paid in terms of performance since the security mechanisms involve encrypting and decrypting data, which take up CPU time.

The authorization service chosen by the client can be either uncertified or certified. In uncertified authorization, the authorization information sent to a server consists only of the client's name. In certified authorization, the authorization information consists of the UUIDs of the client's name and groups. The certified authorization information is in the form of an EPAC, which is produced by the privilege service. In both the certified and uncertified authorization service, the authorization information is sent securely.

The authentication and authorization information about the client are used by the server to determine whether the client should be granted the access to the resource that it has requested. The server knows for certain the identity of the client and what authorization groups the client belongs to. The server can therefore compare the client's credentials against information in ACLs and determine whether a client should be given the access it is requesting.

ACLs

If a distributed application uses ACLs to control access to its resources, then the distributed application programmer needs to write an ACL manager to handle access to the resources. The ACL manager is part of the server side of the application. Typically, there is one ACL associated with each resource that the server manages. The ACL contains one or more entries specifying a user or group and what operations the user or group is permitted to perform on the resource (for example, read, write, or execute permission). The ACL manager takes the authorization information supplied by the application client during an RPC and compares it to the ACL for the requested resource. The ACL manager indicates whether the client is or is not allowed the requested access to the resource.

Figure 31 on page 57 shows a simple DCE ACL. Every DCE ACL contains a field indicating what type of ACL it is. The ACL type in this case is `sp_data_acl`. Each DCE ACL also contains a field indicating what the default cell is for the entries in

the ACL. In the example, the default cell is /.../C=US/O=IBM/OU=DCE. The rest of the ACL consists of ACL entries.

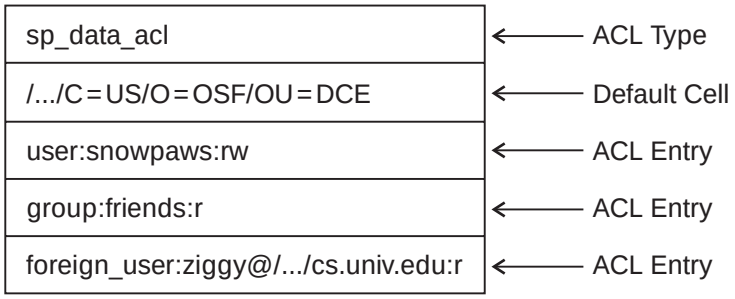

Figure 31. DCE ACL Example

ACL entries can be of several types. The preceding example shows three types of ACL entries: user, group, and foreign_user. The cell to which the user and group entries belongs is the default cell listed in the ACL. The cell to which the foreign_user entry belongs is specified in the entry.

Each entry includes a list of permissions. The different possible permissions are determined by the ACL type (in this example, sp_data_acl). There are two types of permissions in the preceding ACL example: r for read permission, and w for write permission.

Based on this ACL, the sp_data_acl ACL manager will give the principal snowpaws in the cell /.../C=US/O=IBM/OU=DCE read and write permission to the object, the members of the friends group in the /.../C=US/O=IBM/OU=DCE cell read permission to the object, and the principal zig in the foreign cell /.../cs.univ.edu read permission.

DCE Security Service Administration

There are two types of DCE security administration: local and cellwide. The administrator of a DCE machine controls the local passwd_override file. This file determines some security aspects that are specific to the local machine, such as which principals may use the machine, the password for a local account (such as root), and so forth. The local security administrator also controls the local file that contains user and password information, if it exists. (This file may contain a copy of information from the security database to be used in case the security server cannot be reached, or for already existing applications that expect such a local file.) If the machine runs DCE servers that use the audit service (application servers, the DTS server, or the security server) the local security administrator also manages the audit daemon (auditd).

The cell-wide security administrator manages the cell's security server(s). This includes managing the secd process, which provides security services on the security server machine, creating and editing the security database using dcecp, and controlling replication of security data. The cell-wide security administrator can also carry out remote administration of the audit daemons running on hosts in the cell. The cell-wide security administrator is also responsible for administering audit service event numbers and event class numbers to ensure that unique numbers are being issued.

The cell-wide security administrator is also involved in cross-cell authentication. It is possible for clients in one cell to communicate securely with servers in another cell. In order for this to happen, the security administrators in the two cells must register each other's authentication service in their registry. This enables clients in one cell to authenticate to servers in another cell. In this way, it is possible for authorized clients in one cell to access services in a foreign cell.

DCE Security and Kerberos

This section contains a note on the relationship between the DCE Security Service and Kerberos, for those who are already familiar with Kerberos. The DCE authentication service is based on MIT Project Athena's Kerberos Network Authentication Service, Version 5. The Kerberos Key Distribution Center (KDC) server is a part of the DCE security server, `secd`. The authorization information that is created by the DCE privilege server is passed in the Kerberos Version 5 ticket's authorization data field.

The Kerberos user commands `kinit`, `klist`, and `kdestroy` are used in DCE security. The Kerberos API is used internally by DCE security but is not exposed for use by the application programmer. Instead, DCE application programmers use the authenticated RPC API.

The Generic Security Service API

The Generic Security Service (GSS) provides an alternate way of securing distributed applications that handle network communications by themselves. With the Generic Security Service API (GSSAPI), applications that establish the secure connection are like a DCE RPC client. Applications that accept the secure connection are like a DCE RPC server.

The GSS available with DCE includes the standard GSSAPI routines (defined in the Internet RFC 1509), as well as OSF DCE extensions to the GSSAPI routines. These extensions are additional routines that enable an application to use DCE security services.

The GSSAPI combines authentication and authorization under a single security mechanism type. The security mechanism provides applications a choice of either authenticated Kerberos security or authenticated PAC authorization under DCE security.

Although an application that uses GSSAPI may not make explicit calls to RPC routines, the GSSAPI implementation itself uses DCE RPC to communicate with the DCE registry.

Additional Information on DCE Security

For additional information on the DCE Security Service and the GSSAPI, see the following:

1. The security chapters of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Core Components* and the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Core Components* (DCE Security Service only)
2. The `(3sec)` reference pages of the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Reference*

3. The (8sec and 5sec) reference pages of the *IBM DCE Version 3.1 for AIX and Solaris: Administration Commands Reference*

DCE Distributed File Service

Distributed systems can provide many advantages over centralized systems, such as higher availability of data and resources, the ability to share information throughout a very large (even worldwide) system, and efficient use of special computing functionality.

A distributed file system is an example of an application that can take advantage of all of these aspects of a distributed system. It can make files highly available through replication, making it possible to access a copy of a file even if one of the machines on which the file is stored goes down. A distributed file system can provide access to files from anywhere in the world, allowing cooperation among geographically dispersed users. A distributed file system can also give users on machines with very little storage space the ability to access and store data on machines with much more disk space available.

DCE DFS is a distributed client/server application built on the underlying DCE services. It takes full advantage of the lower-level DCE components (such as RPC, the security service, and the directory service). The following subsections describe DFS and the configuration of its components, and they provide various user perspectives on DFS.

What is DFS?

DFS is a distributed application that manages information in the form of a file system. This section describes the units into which DFS data is organized, the active components that manage that data, and the benefits of DFS.

DFS Data Organization

DFS data is organized at three levels. (See Figure 32 on page 60.) The three levels of DFS data are as follows, from smallest to largest:

1. Files and directories

   The unit of user data. Directories organize files (and other directories) into a hierarchical tree structure.

2. Filesets

   The unit of administration. A fileset is a subtree of files and directories that is no larger than a disk or partition (or logical volume, if supported). The fileset is a convenient grouping of files for administrative purposes; for example, the subtree of files pertaining to a particular project can be grouped in the same fileset.

3. Aggregates

   The unit of disk storage, similar to a disk partition. It is also the unit of fileset exporting, which makes the data in filesets available to users of DFS. It can contain one or more filesets.

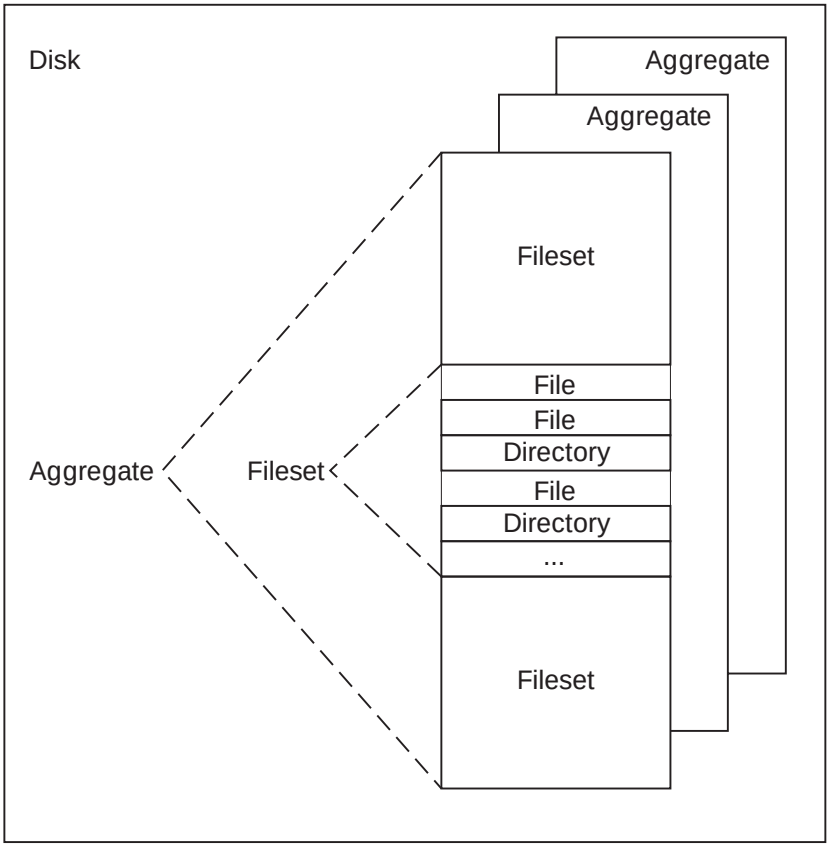

Figure 32. Files, Directories, Filesets, and Aggregates

DFS Components

DFS consists of several components. This section briefly describes each of these components, discussing the software that runs on DCE client machines (the cache manager), then the software that runs on DCE file server machines (the file exporter, token manager, and DCE Local File System), and finally the administrative server processes, which typically run on DFS file server machines (the fileset server, basic overseer server, replication server, update server, fileset location server, and backup server). It also briefly describes the administrative tools used to monitor DFS use and activity (Scout and the `dfstrace` utility), and it describes the DFS/NFS secure gateway, which provides authenticated access to DFS from NFS clients.

Cache Manager: The cache manager is the client side of DFS. The cache manager runs on any machine that is acting as a DFS client. It takes a user's file system request and looks in a local cache to see if a copy of the data is already on the local system. If it does not find the data in the local cache, the cache manager sends a request for the data to the file server machine and caches the data locally, either on disk or in memory.

Because files are cached on the client, a local copy of a cached file can subsequently be accessed instead of the remote copy on the file server machine. As a result, network traffic to the file server machine, as well as file server machine load, is much lighter than if the client had to go to the server each time it needed to access a file.

File Exporter: The file exporter is the server side of DFS. The file exporter runs on a DFS file server machine, where it handles requests from clients for the files that it manages. The file exporter receives an RPC call and accesses its own local file system, which can be the DCE Local File System (DCE LFS) or another file system such as a UNIX File System (UFS), to service the request. Using the token manager, it handles the synchronization of different clients concurrently accessing the same file and returns the requested information to the client.

Token Manager: The token manager runs on a file server machine to synchronize access to files by multiple clients. It does this by issuing tokens, which represent the ability to perform operations. The tokens that a token manager issues to DFS clients carry various access rights, usually read or write. There are four different kinds of tokens: data tokens for access to file and directory data, status tokens for access to file and directory status, lock tokens for locking a portion of a file, and open tokens for opening a file.

The token manager on the server side cooperates with the token management layer in the cache manager (on the client side) to manage tokens. If a client requests an operation that conflicts with a token that another client holds, the token manager must revoke the existing token and grant a new token before the requested operation can proceed.

DCE Local File System: DCE LFS is the physical file system provided with DCE. It manages the storage of files on a disk. The scope of DCE LFS is a single computer. LFS is analogous to a UFS. However, DCE LFS is more powerful than most local UFSs since it includes features that result in greater capabilities than a distributed file service based on a traditional UFS. These capabilities include the ability to use more flexible data protection in the form of DCE ACLs; the ability to replicate, back up, and even move different parts of the file system without interruption to service; and the use of logging for fast recovery after a crash (in contrast to UFSs, which must execute the time-consuming `fsck` command). DCE LFS also includes support for DCE cells; for example, the owner of a file or the name in an entry on an ACL can be a name from a foreign cell.

A UFS can be used as a file server machine's physical file system as an alternative or complement to DCE LFS. DFS can export a UFS, issue synchronization tokens for files in a UFS, and perform fileset operations such as dump and restore on a UFS. However, there is only one fileset per UFS partition, which results in large filesets; and, unlike DCE LFS filesets, UFS filesets cannot be replicated or moved. Although UFS systems are supported in DFS, a file server machine that uses DCE LFS has more functionality than a file server machine that uses only UFS.

Fileset Server: The fileset server allows administrators to create, delete, move, and perform other operations on filesets. For example, the fileset server enables an administrator to move a fileset from one file server machine to another for load balancing. (If DCE LFS is not being used as the physical file system, an entire partition is treated as a single fileset; in this case, some fileset operations may not be supported.)

Basic Overseer Server: The basic overseer server, or BOS server, monitors the DFS processes that run on a server and restarts them when needed. The BOS server maintains information about the processes and responds to administrative requests for that information.

Replication Server: The replication server is an administrative server that handles replication of filesets. For example, an administrator can create read-only copies of

a fileset on multiple file server machines. The replication server updates the replicas either manually, at the request of an administrator, or automatically, as data in the fileset changes. With replication, even if a file server machine that houses one copy of a fileset goes down, another copy of the fileset is still available on another file server machine.

Update Server: The update server provides the ability to distribute binary files or administrative information to machines configured as DFS servers. The update server consists of the `upclient` and `upserver` processes. The `upclient` software runs on a machine that needs to receive new versions of the binary files or administrative information. The `upserver` software runs on a master machine and on request propagates any changes to binaries or administrative information to the machines running the `upclient` software.

Fileset Location Server: The fileset location server, or FL server, provides a replicated directory service that keeps track of the site (file server machine and aggregate) at which each fileset resides. The FL server provides a lookup service analogous to the service CDS provides, with the exception that the FL server is specialized for DFS. It provides fileset location transparency; that is, users can access a fileset simply by knowing its name; they do not need to know the fileset's location. As a result, a fileset can be moved without users and applications being aware of the move. DFS automatically updates the fileset's location in the fileset location database (FLDB).

Backup Server: The backup server is a facility for backing up data on file server machines. The backup server maintains backup records in the replicated backup database. It maintains a schedule for the backing up of file system data, and it has the ability to perform both full and incremental dumps. The unit of backup is the fileset.

Scout: The Scout administrative tool collects and displays information about the file exporters running on file server machines, enabling a system administrator to monitor the use of DFS.

The dfstrace Utility: The `dfstrace` utility allows sophisticated administrators and system developers to trace DFS processes that run in either the user-space or the kernel. The utility consists of a suite of commands that provide low-level diagnostic and debugging information.

NFS/DFS Authenticating Gateway: The NFS/DFS Authenticating Gateway provides authenticated access to DFS from NFS clients. Users who have DCE accounts can authenticate to DCE via a DFS client configured as a gateway server and access DFS data according to their DCE identities. Administrators can allow NFS client users to establish authentication mappings between their NFS client identities and DCE identities.

Some DFS components run in the host machine's kernel. These are the cache manager and token management layer on DFS client machines; and the file exporter, token manager, and DCE LFS on file server machines.

Features of DCE DFS

DCE DFS has the following features:

1. Uniform file access

   DFS is based on a global namespace. A DFS file is accessed by the same name no matter where in the distributed system it is accessed from. Users do not

need to know the network address or name of the file server machine on which the file is located to name and access the file. For example, the file `/.../cs.univ.edu/fs/usr/ziggy/thesis` can be addressed by that name from anywhere in DCE, including from foreign cells.

2. Intracell location transparency

   Data can move from one location to another within a cell without a user or programmer being affected by the move. Because of this transparency, an administrator can move a fileset from one file server machine to another for load balancing, for example, without disturbing users.

3. Performance

   DFS is a high-performance file service. Fast response is achieved in part through the caching of file and directory data on the DFS client machine. This reduces the time it takes for a user to access a file, and it also reduces the traffic on the network and the load on the file server machine. The first time a user on a machine accesses a file, the cache manager gets a copy of the file from the file server machine and caches it on the client machine. Subsequent access to the file can then be made to the copy on the client machine rather than to the copy on the file server machine.

4. Availability

   DFS makes its services and data highly available in several ways. One way is through replication, in which a read-only copy of a file can be stored on more than one file server machine. This way, if the file server machine that houses one copy of the file is down, another copy of the file may still be available on another file server machine. DFS replication is especially useful for files that are accessed by many users but change infrequently (for example, binary files).

   Another way DFS achieves high availability is through caching. Copies of files are cached on DFS clients. Even if a client is temporarily disconnected from the network, users of the client may be able to access copies of files that reside in the local cache.

   DFS administration can occur while users continue to access DFS files, which is another means of providing high availability. Both backups and relocation of DFS filesets can be done without making the data in the filesets unavailable to users.

   The physical file system portion of DFS, DCE LFS, is designed for fast recovery (yielding high availability) after failures. DCE LFS is a log-based file system; that is, DCE LFS keeps a record of actions taken that affect the file system structure so that, in the case of a system crash, the record can be replayed to bring the file system to a consistent state.

5. Support for distributed application programming

   DFS is itself a distributed application, but it in turn supports the development of other distributed applications. Programmers can use DFS to share data or to communicate in a distributed application. DFS takes care of network communications and the movement, synchronization, and storage of shared data.

6. Ease of administration and scalability

   DFS files are grouped into units called *filesets*, which are convenient to administer. The processes that implement DFS, such as the FL server and the backup server, are monitored and maintained automatically by the BOS server, resulting in less work and a more scalable system for a DFS administrator. Because of the high performance mentioned previously, DFS has a high client-to-server ratio. This leads to a scalable system in which clients can be

added with low impact on other clients and the rest of the system. Finally, DFS includes tools such as the update server to automate time-consuming administrative tasks.

7. Integration

   DFS is fully integrated with other DCE components, including RPC, the security service, the directory service, and threads.

8. Interoperation

   DFS interoperates with other file systems; for example, a UFS can be exported to users of DFS.

9. Standards

   DFS maintains POSIX single-site read/write semantics. DCE LFS adheres to POSIX 1003.1.

DFS Configuration

This section describes which of the DFS components run on the different types of DFS machines: DFS client machines, DFS file server machines, and other DFS server machines.

The cache manager runs on every machine that acts as a DFS client. It communicates with file server machines to provide DFS service. (See Figure 33.)

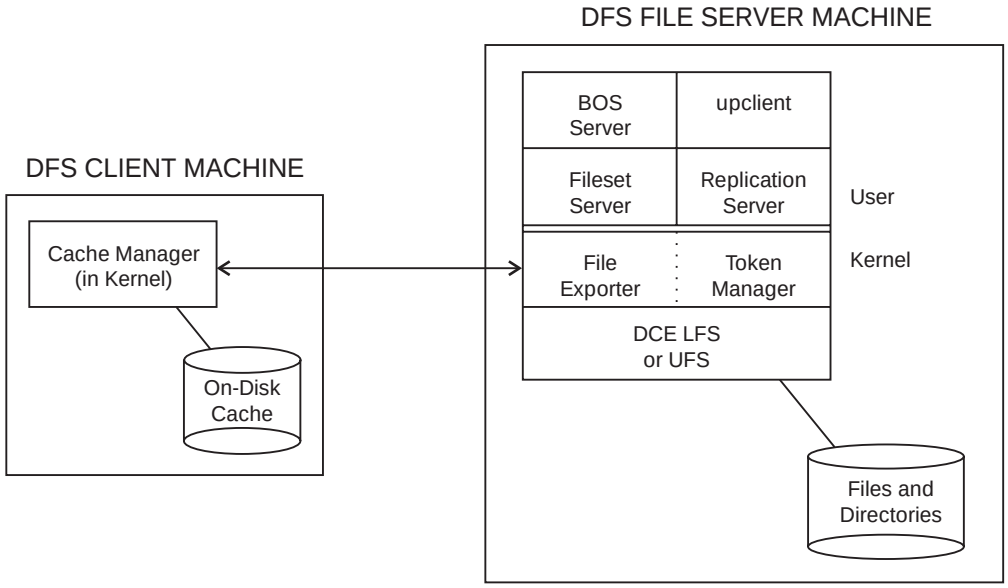

Figure 33. DFS Client and File Server Machines

Several processes run on DFS file server machines: the file exporter (which includes the token manager), the BOS server, the replication server, the fileset server, and the client side of the update server. Also present on the file server machine is a physical file system, DCE LFS, UFS, or both.

Some DFS processes must run on a machine that contains the files or database they access. These processes usually run on DFS file server machines. (See Figure 34 on page 65.)

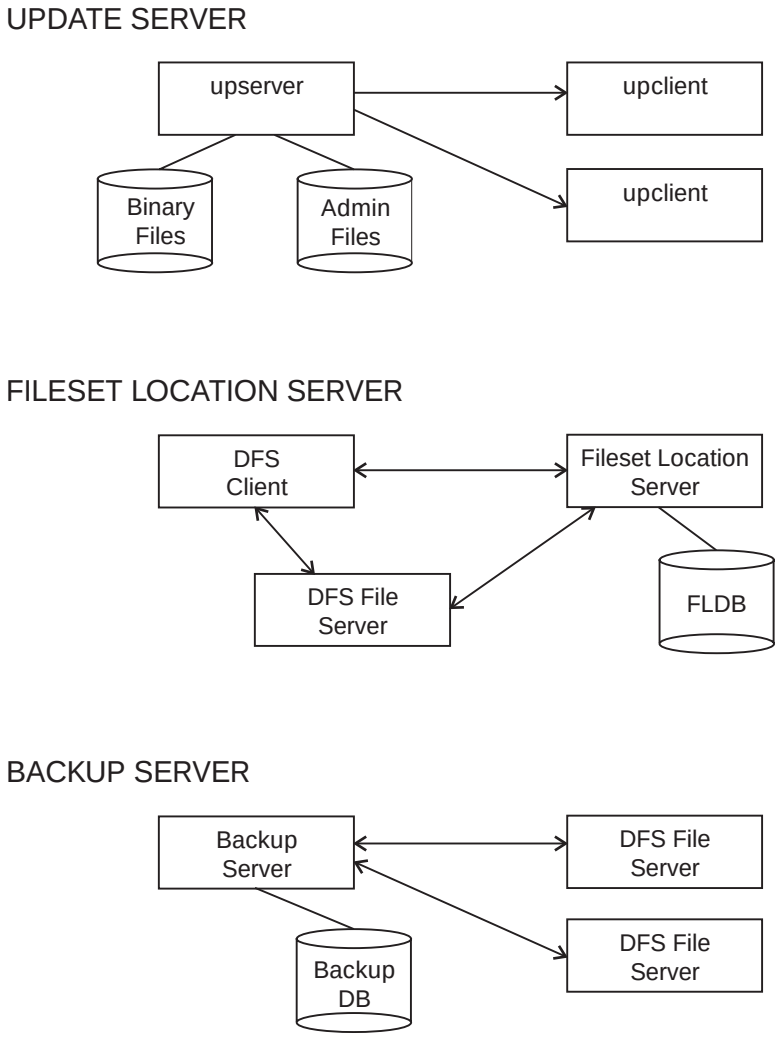

Figure 34. Other DFS Servers

These processes are the server side of the update server (which runs both on machines that contain master copies of configuration files and on machines that contain master copies of binary files), the FL server (which runs on machines on which the fileset location database is located), and the backup server (which runs on machines on which the backup database resides).

End User's Perspective

Users are usually not aware that some of the files that they access are stored on their local computer, some on their cell's file server machines, and some in another cell, because to a user, DCE DFS presents one large, worldwide file system. Users do notice a few differences between working on a distributed file system and working on a local file system. For example, DFS users are issued quotas for file storage, which they can use DFS commands to examine. DFS also includes commands for determining the location of a file and other information that is unique to a distributed file system.

Programming with DFS

Application programmers typically use DFS transparently by making POSIX 1003.1 file system calls. Additional DFS interfaces provide administrative capabilities such as calls for administering filesets. The fact that programmers can use a distributed file system through a familiar interface means that DFS enables distributed applications programming without special distributed programming expertise. Through the use of DFS, programmers can write distributed applications without the use of RPC and the client/server model, assuming the DFS data sharing model is appropriate to the application.

DFS Administration

Administration of DFS is a significant task because several processes that implement DFS need to be set up and maintained. However, administrative tools are provided to aid in this task. DFS configuration is varied and flexible, so a DFS administrator has the additional task of designing and evolving a configuration of DFS servers and clients that best suits the needs of the system's users. DFS day-to-day administration includes fileset administration such as making filesets available, backing them up, and moving them.

Additional Information on DFS

For additional information about the DCE Distributed File Service, refer to your DFS documentation supporting IBM DCE 3.1.

DCE Cross-Component Facilities

For most applications, multiple DCE components work in concert. Several services are dedicated to facilitating interaction among components and are described separately from the components themselves in the following subsections.

Host Services

The DCE host services provide remote system management. Each host runs a DCE host daemon (dced) as the interface to the host services. In many cases dced automatically maintains the data and performs the functions. Some of the data that can be accessed (and maintained) remotely includes the host name, the host's cell name, configuration and execution data for all servers on the host, and a database of endpoints (server addresses) on which running servers can be found. Some of the functions that can be performed remotely include starting and stopping servers.

A security validation service maintains a login context for host identification and certifies for application programs that the DCE security daemon (secd) is legitimate.

The *key table management service* enables remote management of server key tables. A server uses private keys rather than human-readable passwords for authentication. This service can be used to add, remove, and change keys and entire key tables.

The endpoint mapper service maintains a local database (an endpoint map) associating port addresses that locate servers on a host with servers, interfaces, and objects. Remote procedure calls use this service via the RPC runtime to resolve

bindings between clients and servers. The data can be remotely perused, and even changed (although changes to the database are usually performed automatically by dced and the RPC runtime).

Application Message Service

The *application message service* is a general-purpose messages manager for readable character strings that are commonly displayed to application users. The service automatically and transparently takes care of many of the special problems that distributed application messaging can give rise to. The service uses catalog files to maintain message text and explanations separate from the program in a culture- or nationality-specific way.

Serviceability

Serviceability is another kind of message text service with functionality beyond just the display of general-purpose text. To the general-purpose messaging service, Serviceability adds storage of additional attributes specifying subcomponents (program modules), message severity, the action users or programs should take, and the debug level.

Backing Store Databases

DCE provides a backing store library for the convenience of programmers who are writing DCE servers. A *backing store* is a persistent database or persistent object store from which typed data can be stored and retrieved by a key. Designed to satisfy the needs of programmers writing servers that deal with ACLs, this facility can be used to store any data IDL can describe that needs to persist between invocations of applications. The backing store routines can be used in servers, in clients, or in standalone programs that do not involve remote procedure calls.

These cross-component facilities are described in detail in the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Core Components* .

The DCE Control Program

The core services (especially CDS and the security service) for large cells can be complex, with some services being replicated or even partitioned across differing systems. The host services described in the previous section will exist on every computer in the cell. An administrative interface is needed that provides consistent and uniform access to DCE administration functions, wherever they reside, from any and every point in the cell. Administrative commands must work consistently and predictably regardless of the platform on which they execute.

The DCE control program (`dcecp`), first available with IBM DCE Version 2.1, was developed to provide consistent, portable, extensible, and secure access to nearly all DCE administration functions from any point in a DCE cell. The DCE control program implements most of the operations that were previously performed by using various component control programs (for instance `rpccp`, `cdscp`, `rgy_edit`, `acl_edit`, `dtscp`, and `sec_admin`). Where before administrators needed multiple control programs with different syntaxes to perform certain operations such as adding a host to a cell, now only `dcecp` is required. Furthermore, these complex operations can be done now using a single ″task script'' that walks administrators through the pertinent commands prompting for input as necessary.

To do this, dcecp is able to manipulate data (for instance, directories, entries, groups, principals, accounts, and ACLs) stored in the various databases (for instance, the registry, clearinghouses, and ACL managers). It can also perform certain management operations like user create, server disable, and registry synchronize.

The DCE control program is built on a portable command language called Tcl (pronounced ″tickle''), which stands for Tool Command Language. Tcl is a platform-independent command language that runs on every system where IBM DCE Version 2.1 or higher is installed. The Tcl command interpreter is provided along with dcecp. Together, these enable administrators to use variables, if statements, looping functions, and other programming operations to enhance the command set. Administrators can share scripts, moving them to other platforms without change. A common cell environment can be developed by propagating scripts. For information on Tcl, see *Tcl and Tk Toolkit*, by John C. Ousterhout (Addison-Wesley: Menlo Park, CA, 1995).

The DCE control program uses an object-operation syntax, in which an object comes first, followed by an operation. The object-operation order makes it easy to add new objects and operations to DCE.

In summary, dcecp is an interactive command-line interface used to manage most aspects of the DCE core components. Only a few infrequently performed control operations have not been replaced by dcecp.

Two DCE Application Examples

This section presents two implementations of a very simple distributed application called greet. This section assumes some familiarity with UNIX systems and the C programming language. The greet application is implemented two different ways: one using DCE RPC, the other using DCE DFS **(AIX only)**. For a more extensive application example, which uses many more DCE services and facilities, see the timop example in the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide* .

The greet Application: An Implementation Using DCE RPC

This first implementation of the greet application is an example of a simple DCE RPC-based application. The client side of the application sends a greeting to the server side of the application. The server prints the client's greeting and sends a return greeting back to the client. The client prints the server's reply and terminates.

Steps in Developing a DCE RPC Application

This section provides a step-by-step description of the development of the greet application.

1. Generate an IDL template.

   The first step is to run the uuidgen program, which creates a Unique Universal Identifier for uniquely labeling the application's interface. It also creates a template for an IDL file. The following command creates the file greet.idl:

```
uuidgen -i > greet.idl
```

The file greet.idl contains the following:

```
[
uuid(3d6ead56-06e3-11ca-8dd1-826901beabcd),
version(1.0)
]
interface INTERFACENAME
{

}
```

2. Name the interface.

   Replace the string INTERFACENAME in the IDL file with the name of the application interface, in this case, greetif.

```
[
uuid(3d6ead56-06e3-11ca-8dd1-826901beabcd),
version(1.0)
]
interface greetif
{

}
```

3. Define the interface operations.

   Within the braces, write definitions of the operations constituting the interface. In this example, there is only one operation, called greet.

```
/*
 * greet.idl
 *
 * The "greet" interface.
 */

[uuid(3d6ead56-06e3-11ca-8dd1-826901beabcd),
version(1.0)]

interface greetif
{
    const long int REPLY_SIZE = 100;

    void greet(
        [in]            handle_t h,
        [in, string]    char client_greeting[],
        [out, string]   char server_reply[REPLY_SIZE]
    );
}
```

   The first line of the operation definition gives the name of the operation, greet, and indicates by the void declaration that it has no meaningful return value. The next three lines specify the arguments to the operation, namely h, client_greeting, and server_reply. The first argument is a handle containing binding information for the server. The second is a string that is passed from the client to the server (the client's greeting). The third argument is a string returned from the server back to the client (the server's reply).

4. Run the IDL compiler.

   The following command runs the IDL compiler:

```
idl greet.idl
```

   (Some of the commands in this section are somewhat simplified. See "Makefile for the greet Application" on page 74 for the complete command.) Three new files are created automatically as a result of this command:

   a. greet.h

b. greet_cstub.o

c. greet_sstub.o

5. Write the client application code greet_client.c.

   In general, the DCE RPC application programmer writes three application code files:

   a. The client code

   b. The server initialization code

   c. The server operation code

   The following is the client code for the greet application, a file called greet_client.c.

```
/*
 * greet_client.c
 *
 * Client of "greet" interface.
 */

#include <stdio.h>
#include <dce/nbase.h>
#include <dce/rpc.h>

#include "greet.h"
#include "util.h"

int
main(
    int argc,
    char *argv[]
)
{
    rpc_ns_handle_t import_context;
    handle_t binding_h;
    error_status_t status;
    idl_char reply[REPLY_SIZE];

    if (argc < 2) {
        fprintf(stderr, "usage: greet_client <CDS pathname>\n");
        exit(1);
    }

    /*
     * Start importing servers using the name specified
     * on the command line.
     */
    rpc_ns_binding_import_begin(
        rpc_c_ns_syntax_default, (unsigned_char_p_t) argv[1],
            greetif_v1_0_c_ifspec, NULL, &import_context,
&status);
    ERROR_CHECK(status, "Can't begin import");

    /*
     * Import the first server (we could interate here,
     * but we'll just take the first one).
     */
    rpc_ns_binding_import_next(import_context, &binding_h,
&status);
    ERROR_CHECK(status, "Can't import");

    /*
     * Make the remote call.
     */
```

```
    greet(binding_h, (idl_char *) "hello, server", reply);

    printf("The Greet Server said: %s\n", reply);
}
```

In this routine, the client makes two calls to the RPC runtime to acquire binding information needed to communicate with the server. The client then calls the `greet` remote procedure, supplying a greeting to be sent to the server. The client prints the reply received by the server.

6. Write the server initialization code `greet_server.c`.

   The second file that the DCE RPC application programmer must write is the server initialization code. This is *boilerplate* code; that is, it is largely the same for any RPC application. The `greet_server.c` file contains the server initialization code for the `greet` application.

```
/*
 * greet_server.c
 *
 * Main program (initialization) for "greet" server.
 */

#include <stdio.h>
#include <dce/dce_error.h>
#include <dce/rpc.h>

#include "greet.h"
#include "util.h"

int
main(
    int argc,
    char *argv[]
)
{
    unsigned32 status;
    rpc_binding_vector_t *binding_vector;

    if (argc < 2) {
        fprintf(stderr, "usage: greet_server <CDS pathname>\n");
        exit(1);
    }

    /*
     * Register interface with RPC runtime.
     */
    rpc_server_register_if(greetif_v1_0_s_ifspec, NULL, NULL,
&status);
    ERROR_CHECK(status, "Can't register interface");

    /*
     * Use all protocol sequences that are available.
     */
    rpc_server_use_all_protseqs(rpc_c_protseq_max_reqs_default,
&status);
    ERROR_CHECK(status, "Can't use protocol sequences");

    /*
     * Get the binding handles generated by the runtime.
     */
    rpc_server_inq_bindings(&binding_vector, &status);
    ERROR_CHECK(status, "Can't get bindings for server");

    /*
     * Register assigned endpoints with endpoint mapper.
     */
```

```
    rpc_ep_register(
        greetif_v1_0_s_ifspec, binding_vector, NULL,
        (unsigned_char_p_t) "greet server version 1.0", &status);
    ERROR_CHECK(status, "Can't register with endpoint map");

    /*
     * Export ourselves into the CDS namespace.
     */
    rpc_ns_binding_export(
        rpc_c_ns_syntax_default, (unsigned_char_p_t) argv[1],
        greetif_v1_0_s_ifspec, binding_vector, NULL, &status);
    ERROR_CHECK(status, "Can't export into CDS namespace");

    /*
     * Start listening for calls.
     */
    printf("Listening...\n");

    rpc_server_listen(rpc_c_listen_max_calls_default, &status);
    ERROR_CHECK(status, "Can't start listening for calls");

    /*
     * Unregister from endpoint mapper.
     */
    rpc_ep_unregister(
        greetif_v1_0_s_ifspec, binding_vector, NULL, &status);
    ERROR_CHECK(status, "Can't unregister from endpoint map");
}
```

In this file, the server registers its interface with the RPC runtime. It then retrieves the binding information assigned to it by the runtime. It registers its binding information with the RPC endpoint mapper, and then with CDS. It then is ready to service requests. Before exiting, the server unregisters its information in the endpoint map.

7. Write the server operation code `greet_manager.c`.

   The third file that an RPC programmer writes is the code that implements the operations defined in the IDL file. In this case, there is only one operation, `greet`. The `greet_manager.c` file implements this operation.

```
/*
 * greet_manager.c
 *
 * Implementation of "greet" interface.
 */

#include <stdio.h>
#include "greet.h"

void
greet(
    handle_t h,
    idl_char *client_greeting,
    idl_char *server_reply
)
{
    printf("The client says: %s\n", client_greeting);

    strcpy(server_reply, "Hi, client!");
}
```

   The server prints the message it received from the client, then puts its own message in the reply parameter to be sent back to the client.

8. Write any utility code.

In addition to the three standard RPC application code files, greet_client.c, greet_server.c, and greet_manager.c, the greet application contains a utility file for handling errors. This file is called util.c.

```
/*
 * util.c
 *
 * Utility routine(s) shared by "greet" client and server programs.
 */

#include <stdio.h>
#include <dce/nbase.h>
#include <dce/dce_error.h>

void
error_exit(
    error_status_t status,
    char *text
)
{
    unsigned char error_text[100];
    int dummy;

    dce_error_inq_text(status, error_text, &dummy);
    fprintf(stderr, "Error: %s - %s\n", text, error_text);
    exit(1);
}
```

The util.c file comes with a header file called util.h.

```
/*
 * util.h
 *
 * Declarations of utility routine(s) shared by "greet" client
 * and server programs.
 */

#define ERROR_CHECK(status, text) \
    if (status != error_status_ok) error_exit(status, text)

void
error_exit(
    error_status_t status,
    char *text
);
```

9. Compile the client and server programs.

   The greet_client and greet_server programs can now be compiled. The client side of the application is compiled using the following command (again, somewhat simplified):

   ```
   cc -o greet_client greet_client.c \
   greet_cstub.o util.o -ldce
   ```

   The server side of the application is compiled as follows:

   ```
   cc -o greet_server greet_server.c greet_manager.c greet_sstub.o \
   util.o -ldce
   ```

Installing and Running the greet Application

This section describes the process for an administrator who is installing and starting up the greet application, and a user who is running it.

1. Installing the client and server programs

An administrator installs the greet_client program on machines on which users will run the greet application. The administrator also installs the greet_server program on one or more machines that will execute the server part of the greet application.

2. Starting the greet server

   To start up the greet server, the administrator enters the following command on a machine that has the greet server installed:

   ```
   greet_server
   /.../my_cell/subsys/my_company/greet_server
   ```

3. Running the greet application

   To run the greet application, a user enters the following command on any greet client machine:

   ```
   greet_client
   /.../my_cell/subsys/my_company/greet_server
   ```

   The greet server will print the message it received from the greet client. Then the greet client prints the reply that the greet server sent back to it.

Makefile for the greet Application

The commands given in the preceding description for building the greet application have been simplified. Following is the actual Makefile, containing the complete commands for generating the application:

```
DCEROOT = /opt/dcelocal
CC = /bin/cc
IDL = idl
LIBDIRS = -L${DCEROOT}/usr/lib
LIBS = -ldce
LIBALL = ${LIBDIRS} ${LIBS}
INCDIRS = -I. -I${DCEROOT}/usr/include
CFLAGS = -g ${INCDIRS}
IDLFLAGS = -v ${INCDIRS} -cc_cmd "${CC} ${CFLAGS} -c"

all: greet_client greet_server

greet.h greet_cstub.o greet_sstub.o: greet.idl
${IDL} ${IDLFLAGS} greet.idl

greet_client: greet.h greet_client.o util.o greet_cstub.o
${CC} -o greet_client greet_client.o greet_cstub.o \
util.o ${LIBALL}

greet_server: greet.h greet_server.o greet_manager.o util.o \
greet_sstub.o
${CC} -o greet_server greet_server.o greet_manager.o \
greet_sstub.o util.o ${LIBALL}

greet_client.c greet_server.c util.c: util.h
greet_manager.c greet_client.c greet_server.c:
greet.h
```

The greet Application: An Implementation Using DCE DFS

This section describes an implementation of the greet application using DCE DFS. In this version, the client and server use well-known files in the DCE filespace to communicate with each other.

This application looks just like an application that uses a local file system, except for the names of the files in the DCE filespace. The communication (using RPC) is done by DFS and is not visible to the programmer.

Note:

> Please note that this example is intended to be simple, not necessarily to model good programming. For example, a real application would check return values for errors, and would be likely to use the `lock` system call to synchronize client and server access to files, rather than waking up every few seconds to check if a file had been created.

The application contains three files: `dfs_greet.h`, `dfs_greet_client.c`, and `dfs_greet_server.c`.

1. The `dfs_greet.h` file

   This file gives the well-known filenames that the client and server communicate through.

   ```
   /*
    * DCE Program Example Using DFS
    *
    * dfs_greet.h
    */

   #define C_GREET_FILE "/.../my_cell/fs/opt/my_company/greet/client"
   #define S_GREET_FILE
   "/.../my_cell/fs/opt/my_company/greet/server"
   ```

2. The `dfs_greet_client.c` file

   This is the client side of the application.

   ```
   /*
    * DCE Program Example Using DFS
    * dfs_greet_client.c
    *
    * The client writes a message for the server into
    * a well-known file.  It waits until the server has
    * created its own well-known file, then reads the
    * server's message from the file, prints it, and
    * deletes the file.
    */

   #include <stdio.h>
   #include "dfs_greet.h"

   #define C_GREET_TEXT "Hi, server!"

   main()
   {
   FILE *f;
   size_t ret;
   char s[BUFSIZ];

   f = fopen(C_GREET_FILE, "w");
   ret = fwrite(C_GREET_TEXT, sizeof(C_GREET_TEXT), 1, f);
   fclose(f);
   while ((f = fopen(S_GREET_FILE, "r")) == NULL)
   sleep(3);
   ret = fread(s, sizeof(char), BUFSIZ, f);
   fclose(f);
   printf("Server says: %s\n", s);
   unlink(S_GREET_FILE);
   }
   ```

3. The `dfs_greet_server.c` file

This file contains the server side of the greet application.

```
/*
 * DCE Example Program Using DFS
 * dfs_greet_server.c
 *
 * The server waits until the client has created a
 * well-known file, then reads the client's message
 * from the file, prints the message, and removed the
 * file.  The server then writes a message for the
 * client into another well-known file.
 */

#include <stdio.h>
#include "dfs_greet.h"

#define S_GREET_TEXT "Hi, client!"

main()
{
FILE *f;
size_t ret;
char s[BUFSIZ];

while ((f = fopen(C_GREET_FILE, "r")) == NULL)
sleep(3);
ret = fread(s, sizeof(char), BUFSIZ, f);
fclose(f);
printf("Client says: %s\n", s);
unlink(C_GREET_FILE);

f = fopen(S_GREET_FILE, "w");
ret = fwrite(S_GREET_TEXT, sizeof(S_GREET_TEXT), 1, f);
fclose(f);
}
```

The Makefile for creating the client and server programs is as follows:

```
# Makefile for DCE Program Example Using DFS


all: dfs_greet_client dfs_greet_server

dfs_greet_client: dfs_greet.h dfs_greet_client.c
cc -o dfs_greet_client dfs_greet_client.c

dfs_greet_server: dfs_greet.h dfs_greet_server.c
cc -o dfs_greet_server dfs_greet_server.c
```

The greet client and greet server are installed as in the RPC application. They are run in the same way, except they do not take a *servername* argument.

Chapter 4. Integration of DCE Technology Components

One of the advantages of OSF DCE is the integration of its component technologies with one another. Wherever appropriate, DCE technologies make use of other DCE technologies to accomplish their tasks. For example, CDS uses many of the other DCE components—Threads, RPC, DTS, and Security—in providing its service.

Because the DCE technologies are well integrated, they also depend on one another for correct functioning. For example, CDS needs a running DCE security server in order to provide its directory service in a secure manner. These dependencies among technology components have implications for DCE activities such as porting, planning, and bringing up a DCE cell.

This chapter describes how DCE components are integrated and the implications of their resulting interdependencies. First a matrix shows the integration of the technology components. Then a section on each of the components describes its use of other DCE technologies. The final section discusses the impact of technology interdependencies on DCE-related activities.

Integration Matrix

Table 4-1 shows which DCE components are used by each of the other DCE components. The components listed in the leftmost column are the technology consumers. The components listed in the top row are the technology providers. For example, in the box (row RPC, column Threads), the X indicates that RPC makes use of the Threads technology. The abbreviation NA (for Not Applicable) in a box shows the intersection of a technology with itself. A blank box indicates that the consuming technology does not use the providing technology. The following sections include discussions of technology integration, including reasons why certain technologies do not make use of other technologies.

Table 1. DCE Component Integration

	Threads	RPC	CDS	DTS	Security	DFS
Threads	NA					
RPC	X	NA	X		X	
CDS	X	X	NA	X	X	
DTS	X	X	X	NA	X	
Security	X	X	X	X	NA*	
DFS	X	X	X	X	X	NA

Note:

The security service is actually a client of itself, because it uses the audit service to log security service events.

The DCE components support distributed applications, and, in accomplishing that task, they also use each other's services, as shown in the matrix. The use of a given DCE component by another DCE component can provide an example for the application programmer.

Note that many of the boxes are filled in, especially those representing the five most basic components (threads, RPC, CDS, DTS, and security). As a result, some pairs of components have mutual dependencies; for example, the security and CDS components. The security service uses information from CDS, while CDS uses the security service to control access to its information. The implications of these mutual dependencies are discussed in"Implications of Mutual Dependencies" on page 79.

Integration by Technology Component

This section takes each of the DCE technology components in turn and describes its use of other technology components.

1. DCE Threads integration

   The DCE Threads component does not involve distribution across nodes and therefore does not use any other DCE component.

2. DCE RPC integration

   RPC uses threads, CDS, and the security service. Threads are used to allow clients and servers to deal with multiple simultaneous RPCs. Note that, as a result of the use of threads by RPC, any component that uses DCE RPC also uses threads.

   RPC uses CDS to look up servers that support a given interface or object in order to discover the locations of those servers and the protocols that they use.

   RPC uses a notion of time; for example, how long to wait for a reply to a message. However, this involves only the time on the local node, such as comparing the time when a message was sent with the current time to see if a timeout has expired. As a result, RPC does not use DTS timestamps directly. RPC does, however, depend on DTS to help ensure that clocks on different machines run at approximately the same rate.

   The DCE Security Service is used to authenticate the RPC client and server to one another, and to pass authorization information about the client for the server to check against its ACLs.

3. DCE CDS integration

   CDS makes use of several DCE technology components. It uses DCE Threads to allow the CDS server and the CDS clerk to handle multiple requests concurrently. It uses RPC in communications between CDS clerks and CDS servers, as well as in communications between CDS servers, such as for keeping replicated information consistent.

   CDS relies on DTS to maintain synchronized clocks in the network for use in the sequencing of updates to the namespace and for use in replication. CDS uses DCE security's ACLs and authenticated RPC to ensure authorized access to directory data and administrative functions.

4. DCE DTS integration

   DTS uses RPC in the communications between DTS clients and DTS servers. It also uses RPC in the protocol between a time server and a time-provider. Since DTS is based on DCE RPC, which uses DCE Threads, DTS also uses Threads.

   DTS depends on CDS to find time servers and their locations. DTS uses the DCE Security Service to authenticate its interactions and to audit security-critical DTS events.

5. DCE Security Service integration

The DCE security server, like all DCE RPC-based applications, uses DCE Threads. The security server communicates with its clients via DCE RPC. CDS is used to find security servers.

The security service uses a notion of time for the expiration of credentials and for detecting replays of authentication information. It assumes reasonable synchronization of the clocks in the network, which is accomplished in DCE by DTS. The security service does not use DTS timestamps in this version of DCE.

6. DCE DFS integration

   The DFS servers that run in user space (for example, the backup, fileset location, and fileset servers) all use DCE Threads to handle multiple requests. Because the DFS file exporter and cache manager run in the kernel, they do not use DCE Threads; DCE Threads is a user-space, not kernel, threads implementation.

   DFS uses DCE RPC for all remote interaction between the DFS clients (for example, the cache manager and Scout) and servers (for example, the file exporter, fileset location server, and backup server). Because the cache manager and file exporter run in the kernel, they use a kernel version of RPC. DFS uses CDS to locate fileset location servers. DFS uses authenticated RPC and DCE ACLs to protect its resources. DFS relies on DTS to maintain clock synchronization in the network.

Implications of Mutual Dependencies

Mutual dependencies among DCE technology components result in restrictions in areas such as the startup of a cell. For example, since the security service depends on CDS to find the location of a security server, and CDS depends on the security service to verify the authenticity of a CDS server, how can a DCE system ever get started? This section identifies the implications of mutual dependencies in the areas of DCE system startup, porting and testing of DCE, and planning for DCE configuration.

1. Implications for startup

   Mutual dependencies in DCE technologies dictate the order in which some steps must be taken in bringing up a DCE client machine, a DCE server machine, and a DCE cell. In particular, a DCE cell's servers must be started up in a particular order. The security server is started first, since its dependency on CDS can be circumvented through the use of a local file to find security servers. Then the CDS server is started. For information on starting up DCE, see the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Introduction*.

2. Implications for porting and testing

   The interdependencies among DCE technologies constrain the order in which technologies can be ported. DCE Threads can be ported first, because other technologies use it, and it has no dependencies. Many of the other technologies have mutual dependencies, however. To resolve this, a porting effort can proceed by first porting the libraries of all the components, and then going on to port and test the servers. For information on porting DCE technologies, see the *OSF DCE Porting and Testing Guide*.

3. Implications for configuration

   DCE technology interdependencies also have implications for configuration. The servers that other servers depend on are the servers that are the highest priority for replication, in environments where high availability is important. This means that CDS and security servers should be replicated since other DCE servers depend on them in order to operate. Among the various DFS servers,

the fileset location server is the highest priority for replication. For information on DCE configuration, see the *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Introduction*.

4. Implications for application programmers

   Since DCE RPC is integrated with DCE Threads, programmers writing RPC-based applications need to be aware of the implications of using multiple threads of control. See the *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Introduction and Style Guide* and *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Core Components*for information about programming with Threads.

Overview of DCE Documentation

This appendix describes the documentation set supplied with IBM DCE.

DCE Documentation

DCE documentation consists of the following documents:

- *IBM DCE Version 3.1 for AIX: Quick Beginnings*
- *IBM DCE Version 3.1 for Solaris: Quick Beginnings*
- *IBM DCE Version 3.1 for AIX and Solaris: Introduction to DCE*
- *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide*
- *IBM DCE Version 3.1 for AIX and Solaris: Application Development Reference*
- *IBM DCE Version 3.1 for AIX and Solaris: Administration Commands Reference*
- *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide*
- *IBM DCE Version 3.1 for AIX and Solaris: Problem Determination Guide*
- *IBM DCE Version 3.1 for AIX: Release Notes*

A brief description of the purpose and audience of each document follows.

Documentation Intended for Multiple Audiences:

The following documents can be used by audiences of varying levels of technical expertise:

- *IBM DCE Version 3.1 for AIX: Quick Beginnings* provides an overview of DCE, information on the packaging, installation, and configuration of DCE clients and servers, and serves as an introduction to the rest of the DCE documentation.
- *IBM DCE Version 3.1 for Solaris: Quick Beginnings* provides an overview of DCE, information on the packaging , installation, and configuration of DCE clients and servers, and serves as an introduction to the rest of the DCE documentation.
- *IBM DCE Version 3.1 for AIX and Solaris: Introduction to DCE* provides an overview of DCE and serves as an introduction to the remainder of the DCE documentation. It also contains the glossary of terms used in DCE documentation.
- *IBM DCE Version 3.1 for AIX: Release Notes* contains last-minute information that could not be included in the other documents.

Documentation Intended for DCE Administrators:

Guide information for DCE administrators is in the following documents:

- *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide* consists of two volumes:
  - *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Introduction*
  - *IBM DCE Version 3.1 for AIX and Solaris: Administration Guide—Core Components*

  This guide provides conceptual and task-oriented information for the DCE administrator. The first volume is an overview, which describes administering

DCE as a whole, including planning and configuring information. The second volume,*Core Components*, describes management of RPC, CDS, DTS, and Security Service.

Reference information for DCE administrators is in the following document:

- *IBM DCE Version 3.1 for AIX and Solaris: Administration Commands Reference* provides reference material for DCE commands, including administrative commands. It is divided into technology component sections.

Documentation for Application Developers:

Guide information for DCE application developers is in the following documents:

- *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Introduction and Style Guide* provides information describing programming with DCE in general, using its various components and facilities. It also provides information describing relevant issues regarding distributed applications.
- *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Core Components* provides information describing the APIs for the various DCE components.
- *IBM DCE Version 3.1 for AIX and Solaris: Application Development Guide—Directory Services* provides information describing the DCE naming and access to CDS through XDS, the use of the X/Open Directory Service Interface, and the Object Classification Tables. It also contains additional information on XDS and the X.500 Directory.
- *IBM DCE Version 3.1 for AIX and Solaris: Application Development Reference* provides reference material for the DCE programming interfaces. It also has command references for the few commands needed by the DCE programmer, in particular, those used with the RPC component.

List of Acronyms and Abbreviations

This appendix consists of a table that lists the acronyms and abbreviations used in DCE.

Table 2. DCE Acronyms and Abbreviations

Acronym/Abbreviation	Definition
ACF	Attribute Configuration File
ACL	Access Control List
ACSE	Association Control Service Element
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
AT	Attribute Table
AVA	Attribute Value Assertion
BER	Basic Encoding Rules
BOS	Basic OverSeer Server
C	Country
C-ISAM	C-language Indexed Sequential Access Method
CAE	Common Application Environment
CCITT	International Telegraph & Telephone Consultative Committee
CDS	Cell Directory Service
CDSD	Cell Directory Service Daemon
CDSCP	Cell Directory Service Control Program
CDSPI	Cell Directory Service Portable Interface
CPU	Central Processing Unit
CRT	Creation Timestamp
DAP	Directory Access Protocol
DB	Database
DCE	Distributed Computing Environment
DCED	DCE Daemon
DCECP	DCE Control Program
DES	Data Encryption Standard
DFS	Distributed File Service
DIB	Directory Information Base
DIT	Directory Information Tree
DN	Distinguished Name
DNS	Domain Name Service
DSA	Directory System Agent
DSP	Directory System Protocol
DTS	Distributed Time Service
DTSD	Distributed Time Service Daemon
DUA	Directory User Agent
EPAC	Extended Privilege Attribute Certificate
EPV	Entry Point Vector
ERA	Extended Registry Attribute
FIFO	First In, First Out
FLDB	Fileset Location Database
GDA	Global Directory Agent
GDS	Global Directory Service
GECOS	General Electric Common Operating System
GSS	Generic Security Service
IDL	Interface Definition Language

Table 2. DCE Acronyms and Abbreviations (continued)

Acronym/Abbreviation	Definition
IP	Internet Protocol
ISO	International Organization for Standardization
LAN	Local Area Network
LFS	Local File System
LRU	Least Recently Used
NDR	Network Data Representation
NetBIOS	Network Version of Basic Input/Output System
NSAP	Network Service Access Point (OSI)
NSI	Name Service Independent
NTP	Network Time Protocol
O	Organization
OCT	Object Class Taable
OS	Operating System
OS/2	Operating System/2
OSF	Open Software Foundation
OSI	Open Systems Interconnection
OSS	OSI Session Service
OU	Organizational Unit
P	Principal
PAC	Privilege Attribute Certificate
PAG	Process Activation Group
PCS	Portable Character Set
PSAP	Presentation Service Access Point
RDN	Relative Distinguished Name
ROM	Read-Only Memory
ROS	Remote Operation Service
ROSE	Remote Operation Service Elements
RPC	Remote Procedure Call
RPCCP	Remote Procedure Call Control Program
RR	Resource Record (DNS)
RR	Round Robin (scheduling)
SPI	Stub Programming Interface
SRT	Structure Rule Table
TCID	Tape Coordinator ID
TCP/IP	Transmission Control Protocol/Internet Protocol
TDF	Time Differential Factor
TFTP	Trivial File Transfer Protocol
TLI	Transport Layer Interface
TPI	Time Provider Interface
UDP/IP	User Datagram Protocol/Internet Protocol
UFS	UNIX File System
UTC	Coordinated Universal Time
UTS	Update Timestamp
UUID	Universal Unique Identifier
VFS	Virtual File System
WAN	Wide Area Network
XOM	X/Open OSI-Abstract-Data Manipulation
XDS	X/Open Directory Service
XTI	X/Open Transport Interface

Glossary

This Glossary defines terms used in this document and in the remainder of the DCE documentation set. Each term is defined for the audience of the document in which it is found. In some cases, a given term has a different meaning when used in the context of different technology components. This difference is indicated by the technology's abbreviation as a prefix to its definition. For example, the term *server* has a different meaning when used in conjunction with the RPC, CDS, DTS, and DFS technology components. The four definitions are listed in the entry for *server* in the Glossary. When no prefix is given, the definition applies to all DCE documentation.

absolute time.

A point on a time scale. For DTS, absolute time refers to the UTC standard.

abstract class.

GDS: An OM class of OM object of which instances are forbidden. An abstract class typically serves to document the similarities between instances of two or more concrete classes.

Abstract Syntax Notation One (ASN.1).

A notation that both enables complicated types to be defined and also enables values of these types to be specified.

access control list (ACL).

1. Security: Data that controls access to a protected object. An ACL specifies the privilege attribute(s) needed to access the object and the permissions that can be granted, with respect to the protected object, to principals that possess such privilege attribute(s).
2. DFS: The following ACL permissions are defined for file system objects: (1) read (abbreviated `r`): allows you to read a file or, with `x`, list a directory and the ACLs of its objects; (2) write (abbreviated `w`): allows you to modify a file or, with `i`, add a new object to a directory or, with `d`, remove an object from a directory; (3) execute (abbreviated `x`): allows you to execute a file or, with `r`, list a directory and the ACLs of its objects; (4) control (abbreviated `c`): allows you to modify a file's ACLs or a directory's ACLs; (5) insert (abbreviated `i`): with `w`, allows you to add a new object to a directory or, with `w` and `d`, rename an object in a directory; (6) delete (abbreviated `d`): with `w`, allows you to remove an object from a directory or, with `w` and `i`, rename an object in a directory.
3. CDS: The following ACL permissions are defined for CDS: (1) read (abbreviated `r`): allows a principal to look up a name and view the attribute values associated with it; (2) write (abbreviated `w`): allows a principal to change the modifiable attributes associated with a name, except its ACLs; (3) insert (abbreviated `i`): (for use with directory entries only) allows a principal to create new names in a directory; (4) delete (abbreviated `d`): allows a principal to delete a name from the namespace; (5) test (abbreviated `t`): allows a principal to test whether an attribute of a name has a particular value without being able to actually see any of the values (that is, without having read permission to the name). Test permission provides application programs with a more efficient way to verify a CDS attribute value. Rather than reading an entire set of values, an application can test for the presence of a particular value; (6) control (abbreviated `c`): allows a principal to modify the ACL entries associated with a name. Control permission is automatically granted to the creator of a CDS name; (7) administer (abbreviated `a`): (for use with directory entries only) allows a principal to issue `cdscp` commands that control the replication of directories.
4. GDS: A recurring attribute of an entry for specifying the access authorization for an object. The following ACL permissions are defined for GDS: (1) `MODIFY PUBLIC`: specifies the user, or subtree of users, that can modify attributes classified as public attributes; (2) `READ STANDARD`: specifies the user, or subtree of users, that can read attributes classified as standard attributes; (3) `MODIFY STANDARD`: specifies the user, or subtree of users, that can modify attributes classified as standard attributes; (4) `READ SENSITIVE`: specifies the user, or subtree of users, that can read attributes classified as sensitive attributes; (5) `MODIFY SENSITIVE`: specifies the user, or subtree of users, that can modify attributes classified as sensitive attributes.

access control list entry.

Data in an ACL that specifies a set of permissions. In the case of a principal or group entry, the permission set is that which can be granted to a principal having the privilege attribute specified in the entry; in the case of a mask entry, the permission set is that which masks the permission set in a principal or group entry.

access control list facility.

A DCE security facility that enables a principal's access to an object to be determined by a comparison of the principal's privileges to entries in an object's ACL.

access right.

See permission.

accessible.

Said of an object for which the client possesses a valid designator or handle.

account.

An entry in the registry database that defines a principal's network identity by associating the principal with a group and optional organization, and with related account information such as the password used to authenticate a principal's identity.

ACF.

See attribute configuration file.

ACL.

See access control list.

active context handle.

RPC: In RPC applications, a context handle that the remote procedure has set to a nonnull value and passed back to the calling program; the calling program supplies the active context handle in any future calls to procedures that share the same client context. *See also* client context, context handle.

address.

An unambiguous name, label, or number that identifies the location of a particular entity or service. *See also* presentation address.

administration domain.

GDS: A collection of several DSAs that share the same schema object (mastered by one of these DSAs and shadowed by all the others).

administrative domain.

1. DFS: A collection of machines configured as the server machines necessary to be administered as a single unit. The administration is typically handled by groups of administrative users.
2. GDS: A collection of several DSAs that share the same schema object (mastered by one of these DSAs and shadowed by all the others).

administrative list.

DFS: A file used to determine who can issue commands that affect filesets or DFS server processes. Administrative lists allow system administrators to control the security of the administrative domains in a cell. *See also* administrative domain, privilege required.

aggregate.

DFS: A logical unit of disk storage that can contain multiple DCE LFS filesets or a single UFS fileset. An aggregate is physically equivalent to a standard UNIX disk partition, but a DCE LFS aggregate supports an optimized metadata structure and a number of specialized fileset-level operations not available on standard UNIX partitions. A UFS partition exported into the global namespace is referred to as an aggregate even though it does not support the optimizations and features of a DCE LFS aggregate.

aggregate identifier.

DFS: The part of the fileset representation that identifies the aggregate on the file server machine on which the fileset is stored.

alias.

1. GDS: A name for a (directory) object, provided by the use of one or more alias entries in the DIT.
2. Security: An optional alternate name for a primary name in the registry database. Aliases and the primary name for which they are an alternate share the same UUID and UNIX ID.

alias entry.

GDS: A directory entry, of object class alias, containing information used to provide an alternative name for an object.

aliased object.

The object to which an alias entry refers.

aliasing.

RPC: Occurs when two pointers of the same operation point at the same storage.

anode.

DFS: An abstraction for referring to an open-ended address space of storage. *See also* vnode.

anonymous user.

A user who is not entered in the directory as an object and who logs into the directory service without giving a name and password.

API.

See application programming interface.

application programming interface (API).

A set of runtime routines or system calls that allows an application program to use a particular service provided by either the operating system or another application program.

application thread.

RPC: A thread of execution created and managed by application code. *See also* client application thread, local application thread, RPC thread, server application thread.

ASN.1.

See Abstract Syntax Notation One.

asynchronous operation.

An operation that does not of itself cause the process requesting the operation to be blocked from further use of the CPU. This implies that the process and the operation are running concurrently.

AT.

See attribute table.

at-most-once semantics.

RPC: A characteristic of a procedure that restricts it to executing once, partially, or not at all—never more than once. *See also* idempotent semantics, broadcast semantics, maybe semantics.

atomic transaction.

DFS: A transaction that happens entirely or not at all; used when partial completion of a transaction is undesirable.

attention threshold.

DFS: In the scout program, the value at which the program highlights a statistic in its graphical display. Separate attention thresholds can be set for most scout statistics. *See also* scout.

attribute.

1. Threads: The individual components of the attributes object. Attributes specify detailed properties about the objects to be created.
2. RPC: (1) An IDL or ACF syntax element, occurring within [] (brackets), and conveying information about an interface, type, field, parameter, or operation. (2) An attribute of an entry in a name service database that stores binding, group, object, or profile information for an RPC application and identifies the entry as an RPC server entry; an NSI attribute.
3. DTS: A piece of information associated with a DTS entity or command. DTS has four attribute categories: characteristics, counters, identifiers, and status.
4. XDS: Information of a particular type concerning an object and appearing in an entry that describes the object in the DIB.
5. XOM: A component of an object, comprising an integer that denotes the attribute's type and an ordered sequence of one or more attribute values, each accompanied by an integer denoting the value's syntax.

attribute configuration file (ACF).

RPC: A .acf file. An optional companion to an interface definition file (a .idl file) that modifies how the DCE IDL compiler locally interprets the interface definition. *See also* interface definition, Interface Definition Language.

attribute configuration language.

RPC: A high-level declarative language that provides syntax for attribute configuration files. *See also* attribute configuration file.

attribute encoding type.

A specifier of the data format (for example, integer, string, UUID) of an attribute value.

attribute instance.

An attribute type UUID and value created according to the attribute type's semantics and attached to a registry object. (Also called attribute or ERA.)

attribute schema.

A collection of attribute type definitions or schema entries. (Also called schema.)

attribute schema object.

See schema object.

attribute set.

An attribute instance with encoding type attr_set. Its value is a list of attribute type UUIDs that identify member attributes of this set. Attribute sets are created for the purpose of efficient queries for related attributes.

attribute syntax.

GDS: A definition of the set of values that an attribute can assume. It includes the data type, in ASN.1, and usually one or more matching rules by which values can be compared.

attribute table (AT).

GDS: A recurring attribute of the directory schema with the description of the attribute types that are permitted.

attribute type.

1. XDS: The component of an attribute that indicates the class of information given by that attribute. It is an object identifier, so it is completely unique.
2. XOM: Any of the various categories into which the client dynamically groups values on the basis of their semantics. It is an integer unique only within the package.
3. Security: The description of the identifiers (such as name and UUID) and semantics (such as encoding type and access control parameters) of instances of this type.

attribute value.

1. XDS: A particular instance of the class of information indicated by an attribute type.
2. XOM: An atomic information object.
3. Security: The data in an attribute instance.

attribute value assertion (AVA).

GDS: A proposition, which may be true, false, or undefined, concerning the values (or perhaps only the distinguished values) of an entry.

attribute value syntax.

See attribute syntax, syntax.

audit action.

A component of the filter directive that specifies where the audit record is to be written: to the console or to an audit trail file.

audit client.

Users of the DCE Audit Service. All DCE servers and user-written distributed applications can be audit clients.

audit condition.

A component of the filter directive that specifies the required outcome of the event before an audit record is written to the audit trail file.

audit daemon.

A DCE component. It maintains the audit filters and the central audit trail file.

audit event.

An occurrence in the use of the application that requires logging of audit records. Generally, audit events involve the integrity of the system.

audit filter.

Used to narrow down the conditions by which audit records are logged. A filter provides a means to specify these conditions.

audit record.

Contains information pertaining to an audit event.

audit trail file.

A set of audit records that provide evidence of the sequence of events that occurred on the system.

authentication.

The verification of a principal's network identity.

authentication header.

A record containing a ticket and an authenticator to be presented to a server as part of the authentication process.

authentication level.

See protection level.

authentication path.

The sequence of cells transited when a principal in one cell communicates with one in another cell. Also known as a trust path.

authentication protocol.

A formal procedure for verifying a principal's network identity; Kerberos is an instance of a shared-secret authentication protocol.

authentication service.

One of the services provided by DCE Security: the authentication service authenticates principals according to a specified authentication protocol. *See also* authentication protocol.

authentication surrogate.

A type of principal represented by an entry in a cell's registry that specifies the same secret key as a corresponding entry in another cell's registry. The authentication services of the two cells use the secret key for the purpose of exchanging data about principals without either authentication service having to share its private key with the other. Authentication surrogates are necessary for intercell authentication. *See also* peer trust.

authenticator.

A record containing information that can be shown to have been recently generated via a conversation key known only by two principals that are participating in an authenticated network exchange.

authorization.

1. The determination of a principal's permission(s) with respect to a protected object.
2. The approval of a permission sought by a principal with respect to a protected object.

authorization data.

That portion of a Kerberos ticket that contains data necessary for authorization decisions. Sometimes abbreviated Auth_Data or A_D.

authorization protocol.

A formal procedure for establishing the authorization of principals with respect to protected objects. Authorization protocols supported by DCE Security include one based on PACs and EPACs (DCE authorization) and one based on names (name-based authorization) *See also* PAC, EPAC, name-based authorization.

automatic binding method.

RPC: A method of managing the binding for a remote procedure call. The automatic method completely hides binding management from client application code. If the client makes a series of remote procedure calls, the stub passes the same binding handle with each call. *See also* binding handle, implicit binding method, explicit binding method.

AVA.

See attribute value assertion.

background skulk time.

An automatic timer that guarantees a maximum lapse of time between skulks of a CDS directory, regardless of other factors, such as namespace management activities and user-initiated skulks. Every 24 hours, a CDS server checks each master replica in its clearinghouse and initiates a skulk if changes were made in a replica since the last time a skulk of that replica completed successfully.

backup.

DFS: The dump of a fileset to a permanent medium such as tape. To back up also means to clone a read/write fileset, which results in a backup fileset.

backup database.

DFS: A database that records the dump schedule for backups, the backup system's tape coordinators, the fileset families that can be dumped, and other administrative information.

backup database machine.

DFS: A server machine in a cell that houses the backup database. *See also* server machine.

backup fileset.

DFS: A fileset created by cloning (copying) a read/write fileset (referred to as the source fileset). The backup version always resides on the same aggregate as its source and usually requires little disk space. It preserves the state of the read/write fileset at the time of the cloning. *See also* clone, read-only fileset, read/write fileset.

backup fileset ID.

DFS: A unique fileset identification number (fileset ID) assigned to the backup version of a fileset.

backup server.

DFS: A server process that runs on backup database machines (which house the backup database). It communicates with the backup database to back up and restore filesets and aggregates.

backup system.

DFS: A system that allows you to copy fileset data to tape and restore it from tape if necessary. The DFS backup system consists of the backup server, the backup database, and one or more tape coordinator machines. *See also* dump, restore.

basename.

DFS: In the scout program, the DCE pathname prefix common to the file server machines to be monitored. If specified on the command line, the basename is displayed in the program's banner line. *See also* scout.

Basic Encoding Rules (BER).

A set of rules used to encode ASN.1 values as strings of octets.

basic overseer server (BOS server).

DFS: A server process that runs on all DFS server machines. It monitors the other DFS server processes running on its machine; it can usually restart those that fail without requiring intervention from a human operator.

BER.

See Basic Encoding Rules.

big endian.

An attribute of data representation that reflects how multioctet data is stored in memory. In big endian representation, the lowest addressed octet of a multioctet data item is the most significant. *See also* endian, little endian.

binary distribution machine.

DFS: A server machine that distributes DFS binaries to other file server machines of its machine type (same CPU/operating system). It runs the server portion of the update server for this purpose. There is one binary distribution machine of each machine type that the cell uses as a DFS server machine. *See also* server machine, update server, upserver.

binary timestamp.

An opaque 128-bit (16-octet) binary number that represents a DTS time value.

binding.

RPC: A relationship between a client and a server involved in a remote procedure call.

binding handle.

RPC: A reference to binding information that defines one possible binding (a client/server relationship). *See also* binding, customized binding handle, primitive binding handle.

binding handle vector.

RPC: A data structure that contains an array of binding handles and the size of the array. *See also* binding handle.

binding information.

RPC: Information about one or more potential bindings, including an RPC protocol sequence, a network address, an endpoint, at least one transfer syntax, and an RPC protocol version number. *See also* binding, endpoint, network address, RPC protocol sequence, RPC protocol, transfer syntax.

binding management method.

RPC: Any of the methods for managing the binding for a remote procedure call. *See also* automatic binding method, implicit binding method, explicit binding method.

blocking call.

A call in which a caller is suspended until a called procedure completes.

bnode.

DFS: A structure that describes common characteristics of the BOS server process. There are two types: simple and cron. Processes are created through bnodes. *See also* basic overseer server.

BOS server.

See basic overseer server.

broadcast.

Threads: To wake all threads waiting on a condition variable. *See also* signal.

broadcast semantics.

RPC: A form of idempotent semantics that indicates that the operation is always broadcast to all host systems on the local network, rather than delivered to a specific system. An operation with broadcast semantics is implicitly idempotent. Broadcast semantics are supported only by connectionless protocols. *See also* at-most-once semantics, idempotent semantics, maybe semantics.

browser.

A Motif-based program that lets users view the contents and structure of a cell namespace.

butc process.

DFS: A process that runs on a tape coordinator machine to monitor the activity of a tape drive. One butc process must run for each tape drive on the machine. *See also* tape coordinator.

C interface.

The interface, defined at a level that depends on the variant of C standardized by ANSI.

C-stub.

The part of the DUA that implements the connection with the communications network.

cache.

1. CDS: The information that a CDS clerk stores locally to optimize name lookups. The cache contains attribute values resulting from previous lookups, as well as information about other clearinghouses and namespaces. The cache is written to disk periodically so that it can survive a system reboot. *See also* copy.
2. DFS: A reserved amount of disk or memory space on a DFS client machine. The DFS cache manager uses the cache to temporarily store files or parts of files retrieved from DFS file server machines so that future access time and network load are reduced. DFS uses a cache-consistency mechanism (token-passing) to guarantee that the source and cached data are consistent. *See also* caching.

cache manager.

DFS: The portion of a DFS client machine's kernel that communicates with DFS server processes by translating local file requests into RPCs (if needed). It stores the requested files in a local disk or memory cache, from which it makes the files available to users on that machine.

caching.

DFS: The technique of copying a file from a file server machine (its central storage place) to a client machine's local disk or memory; users then access the copy locally. Caching reduces network load because a file does not have to be fetched across the network more than once (unless the central copy changes).

caching layer.

DFS: The part of the DFS cache manager that manages the cached data, performing fetches and stores and answering status requests.

call chain.

The chain of operations (RPC calls) leading from the delegation initiator to the final target.

call queue.

RPC: A first-in, first-out queue used by an RPC server to hold incoming calls when the server is already executing its maximum number of concurrent calls.

call thread.

RPC: A thread created by a server's RPC runtime to execute remote procedures. When engaged by a remote procedure call, a call thread temporarily forms part of the RPC thread of the call. *See also* application thread, RPC thread.

callback.

DFS: A procedure that is registered with a token to be called automatically if the token is revoked. The act of revoking a token is also referred to as a callback.

cancel.

1. Threads: A mechanism by which a thread informs either itself or another thread to terminate as soon as possible. If a cancel arrives during an important operation, the canceled thread may continue until it can terminate in a controlled manner.

2. RPC: A mechanism by which a client thread notifies a server thread (the canceled thread) to terminate as soon as possible. *See also* thread.

CDS.

See DCE Cell Directory Service.

CDS Advertiser. *See* Cell Directory Service Advertiser.

CDS control program (cdscp).

A command interface that CDS managers use to control CDS servers and clerks and manage the namespace and its contents.

CDS-defined attribute.

A standard attribute that CDS associates with names. A specific CDS-defined attribute has the same meaning no matter what type of entry (clearinghouse, directory, object) it is associated with. However, different types of entries can have different CDS-defined attributes. For example, every CDS name has the CDS-defined attributes of Creation Timestamp (CDS_CTS), Update Timestamp (CDS_UTS), and Access Control Set (CDS_ACS). In addition to those attributes, a soft link has unique CDS-defined attributes containing its expiration time and the name it points to.

cdscp.

See CDS control program.

cell.

1. The basic unit of operation in DCE. A cell is a group of users, systems, and resources that are typically centered around a common purpose and that share common DCE services. At a minimum, a cell configuration includes one cell directory server, one security server, and one distributed time server. A cell can consist of from one system to as many as several thousand systems. Systems in the cell can be in the same geographic area (for example, on the same LAN), but geography does not necessarily determine a cell's boundaries. The boundaries of a cell are typically influenced by its purpose, as well as by security, administrative, and performance considerations. With respect to individual DCE technologies, a cell represents the following definitions.
2. CDS: A unified naming environment consisting of CDS clerks and servers.
3. DFS: An administratively independent installation of server and client machines.
4. Security: The set of principals that share their secret keys with the same authentication service.

cell alias.

DFS: An additional global name given to a cell.

Cell Directory Service (CDS) Advertiser. A daemon that starts a CDS clerk, when needed, to access the CDS name space. On the CDS server machine, the advertiser broadcasts the existence of the server every 10 minutes and each time a client is first started. On the client machine, the advertiser receives broadcasts from the CDS server which inform the clients of the existence of the CDS servers and their addresses.

cell module.

DFS: The part of the DFS cache manager that maintains a list of cells that have been contacted.

cell-relative name.

See local name.

central audit trail file.

The audit trail file that is maintained by the audit daemon. This is created and used if the user does not specify an audit trail file when starting the audit daemon.

chaining.

A mode of interaction optionally used by a DSA that cannot perform an operation itself. The DSA chains by invoking an operation of another DSA and then relaying the outcome to the original requester.

character set.

A group of characters, such as the English alphabet, Japanese Kanji, and the European character set.

characteristic attribute.

A type of attribute that reflects or affects the behavior of a software entity. You generally can set or change characteristic attributes.

child cell.

A cell whose name is stored in the CDS server of another cell (its parent cell) and includes its parent cell's name as a prefix to its own name.

child directory.

A CDS directory that has a directory above it is considered a child of the directory immediately above it.

child pointer.

A pointer that connects a directory to a directory immediately below it in a namespace. You do not explicitly create child pointers; CDS creates them for you when you create a new directory. CDS stores the child pointer in the directory that is the parent of the new directory.

ciphertext.

The output of an encryption function. Encryption transforms plaintext into ciphertext.

class.

A category into which objects are placed on the basis of both their purpose and their internal structure. *See also* object class, OM class.

class-id.

A component of the event class number, which identifies the event class within the set of event classes.

class-specific attribute.

CDS: An attribute that has meaning only to a particular class of object and to the application using that object class. A CDS object's class can be defined in an attribute named CDS_Class. Programmers who write applications that use CDS can define their own object classes and class-specific attributes.

clearinghouse.

A collection of directory replicas on one CDS server. A clearinghouse takes the form of a database file. It can exist only on a CDS server node; it cannot exist on a node running only CDS clerk software. Usually only one clearinghouse exists on a server node, but there may be special cases when more than one exists.

clearinghouse object entry.

A special class of object entry that describes a clearinghouse. The clearinghouse object entry is a pointer to the network address of an actual clearinghouse. This pointer enables CDS to find a clearinghouse and use and manage its contents. A clearinghouse modifies and manages its own object entry when necessary; normally CDS managers do not need to maintain it. The clearinghouse object entry has the same name as the clearinghouse.

clerk.

1. CDS: The software that provides an interface between client applications and CDS servers. The clerk receives a request from an application, sends the request to a CDS server, and returns any resulting information to the application. The clerk saves (caches) the results of lookups so that it does not have to repeatedly go to a CDS server for the same information.
2. DTS: A software component that synchronizes the clock for its client system by requesting time values from servers, computing a new time from the values, and supplying the computed time to client applications.

client.

1. CDS: Any application that interacts with a CDS server through the CDS clerk.
2. DTS: Any application that interacts with a DTS server through the DTS clerk.
3. RPC: The party that initiates a remote procedure call. Some applications act as both an RPC client and an RPC server. *See also* server.
4. DFS: A consumer of resources or services. *See also* server.
5. GDS: Consists of an application that links the DUA library, the C-stub that handles the connection over the communications network for accessing a remote server, and the DUA cache.

client application thread.

RPC: A thread which is executing client application code that makes one or more remote procedure calls. *See also* application thread, local application thread, RPC thread, server application thread.

client binding information.

RPC: Information about a calling client provided by the client runtime to the server runtime, including the address where the call originated, the RPC protocol used for the call, the requested object UUID, and any client authentication information. *See also* binding information, server binding information.

client context.

RPC: The state in an RPC server's address space generated by a set of remote procedures (manager) and maintained across a series of calls for a particular client. *See also* manager, context handle.

client machine.

DFS: A machine whose kernel includes the DFS cache manager. A client machine is capable of requesting data from remote file exporters and caching the data locally. *See also* server machine.

client portion of update server.

See upclient.

client stub.

RPC: The surrogate code for an RPC interface that is linked with and called by the client application code. In addition to general operations such as marshalling data, a client stub calls the RPC runtime to perform remote procedure calls and, optionally, manages bindings. *See also* server stub, stub.

clock.

The combined hardware interrupt timer and software register that maintain the system time. In many systems, the hardware timer sends interrupts to the operating system; at each interrupt, the operating system adds an increment to a software register that contains the time value.

clock adjustment.

DTS: Process of changing the system clock time by modifying the incremental value that is added to the clock's software register for a specified duration.

clone.

DFS: A backup or read-only copy of a fileset created by copying only the read/write (source) fileset's header rather than the data it contains. The clone preserves pointers to fileset data that existed when the clone was made; it therefore must exist on the same aggregate as the source. Cloning a fileset also refers to making a copy of it with the proper `fts` commands for later use with the DFS backup system. *See also* `replica`.

clone ID number.

DFS: The fileset ID number of the last clone made from the fileset's read/write source for the purpose of replication.

code point.

Location in the distributed application code that designates the operations in the application where logging of audit records may be required. The DCE audit APIs are called in the application's code points.

code set.

The mapping of the members of a character set to specific numeric code values. Examples of code sets include ASCII, JIS X0208, and ISO 8859-1.

code set registry.

A per-host file that contains, for each code set supported on the host, a mapping between a string name for the code set (which is the name used on the host to refer to the code set) and the unique identifier that has been assigned (by OSF or by the site) to the code set.

collapse.

To remove the contents of a directory from the display (close it) via the CDS browser. To collapse an open directory, you double-click on its icon. Double-clicking on a closed directory expands it.

command suite.

DFS: The DFS command suites are `bak`, `bos`, `cm`, `dfsgw`, `dfstrace`, and `fts`.

commit.

DFS: An indication that all of the actions associated with a specific transaction have been written to the log. Once a transaction has committed, its actions are permanent. In the event of system problems, those actions are repeated when the system's recovery mechanism replays the log.

communications link.

RPC: A network pathway between an RPC client and server that uses a valid combination of transport and network protocols that are available to both the client and server RPC runtimes.

compatible server.

RPC: A server that offers the requested RPC interface and RPC object and that is available over a valid combination of network and transport protocols that are supported by both the client and server RPC runtimes.

computed time.

The result of the synchronization process—the time value that the clerk or server process computes according to the values it receives from several servers.

concrete class.

An OM class of which instances are permitted.

condition variable.

A synchronization object used in conjunction with a mutex. A condition variable allows a thread to block until some event happens.

configuration of directory service.

GDS can be configured as a client system or a client/server system. In a client system, a DUA either accesses the local DUA cache or a remote server over the communications network. In a client/server system, a DUA either accesses a local server or a remote server over the communications network. The local server is also accessible from a remote client or server.

conformant array.

RPC: An array whose size is determined at runtime. A structure containing a conformant array as a field is a conformant structure.

connection-oriented protocol.

A connection-based, reliable, virtual-circuit transport protocol, such as TCP; an RPC protocol that runs over a connection-based transport protocol.

connectionless. Not connected. For example, a connected datagram is a network protocol that is connected. A connectionless datagram network protocol is one that is not connected.

container.

Containers are objects that hold other objects. The objects they hold can themselves be either simple objects or container objects. Simple objects do not hold other objects. Files are simple objects, and directories are containers. The directories can hold simple objects (files) and other containers (subdirectories). *See also* `container object`, `simple object`.

container object.

An object that can hold another object. For example, a directory is a container object since it can hold files. *See also* `simple object`.

context handle.

RPC: A reference to the state (client context) maintained across remote procedure calls by a server on behalf of a client. *See also* client context.

continuation reference.

Describes how the performance of all or part of an operation can be continued at a different DSA or DSAs. *See also* referral.

control access.

CDS: An access right that grants users the ability to change the access control on a name and do other powerful management tasks, such as replicate a directory or move a clearinghouse.

convergence.

The degree to which CDS attempts to keep all replicas of a directory consistent. Two factors control the persistence and speed at which CDS keeps directory replicas up to date: the setting of a directory's CDS_Convergence attribute and the background skulk time. You can set the CDS_Convergence attribute to high, medium, or low. By default, every directory inherits the convergence setting of its parent. *See also* background skulk time.

conversation key.

A short-lived encryption key provided by the authentication service to two principals for the purpose of ensuring secure communications between them.

Coordinated Universal Time (UTC).

An international time standard that DTS uses. The zero hour of Coordinated Universal Time is based on the zero hour of Greenwich (England) Mean Time.

copy.

GDS: Either a copy of an entry stored in other DSAs or a locally and dynamically stored copy of an entry resulting from a request (a cache copy).

core leak.

DFS: A situation that can develop as a process allocates virtual memory but does not free it again. When memory is completely exhausted, the machine crashes. The BOS server can be configured to restart all processes on a file server machine once a week to reduce the likelihood of core leaks.

courier.

DTS: A local server that requests a time value from a randomly selected global server each time it synchronizes.

Creation Timestamp (CTS).

An attribute of all CDS clearinghouses, directories, soft links, child pointers, and object entries that contains a unique value reflecting the date and time the name was created. The timestamp actually consists of two parts: a time portion, and a portion containing the system identifier of the node on which the name was created. This guarantees uniqueness among timestamps generated on different nodes.

credentials.

A general term for privilege attribute data that has been certified by a trusted privilege certification authority. The DCE authorization protocol implements credentials as Privilege Attribute Certificates (PACs).

cron bnode.

DFS: A bnode that manages a single process that is to be run either exactly once or periodically. *See also* basic overseer server, bnode.

cron process.

DFS: A type of process defined in a server machine's BosConfig file. It executes weekly or daily at a defined time rather than running continuously. *See also* cron bnode, simple process.

CTS.

See Creation Timestamp.

customized binding handle.

RPC: A user-defined data structure from which a primitive binding handle can be derived by user-defined routines in application code. *See also* primitive binding handle.

daemon. A program that runs unattended to perform a standard service. Some daemons are triggered automatically to perform their tasks; others operate periodically. An example is the **cron** daemon, which periodically performs the tasks listed in the **crontab** file.

DAP.

See Directory Access Protocol.

Data Encryption Standard (DES).

A data encryption algorithm widely used in the United States.

data limit.

RPC: A value that specifies which elements of an array are transmitted during a remote procedure call.

data token.

DFS: A token that grants access to a range of bytes in a file. Read and write data tokens are available. *See also* token.

datagram.

An unreliable network data packet that is independent of all other packets and lacks any guarantees of delivery or sequentiality.

datagram protocol.

A connectionless, datagram-based transport protocol, such as UDP; an RPC protocol that runs over a connectionless transport protocol.

date-specific restore.

DFS: In the DFS backup system, a restore that returns a fileset to its state when it was last dumped before a specified date. A date-specific restore differs from a full restore. *See also* full restore, restore.

DCE.

See Distributed Computing Environment.

DCE Audit Service.

That part of the DCE Security Service which detects and records the execution of DCE server operations that are relevant to the maintenance of a secure distributed computing environment. *See also* DCE Security Service.

DCE authorization.

Provides a server with the client's PAC and EPAC. *See also* PAC, EPAC, name-based authorization, authorization protocol.

DCE Cell Directory Service (CDS).

The DCE Cell Directory Service stores names and attributes of resources located in a DCE cell. It is optimized for local access, since most directory service queries are for information about resources within the same cell as the originator of the query. It is replicated, in order to make it highly available. There must be at least one cell directory server in each DCE cell.

DCE control program (dcecp).

An administrative interface that provides consistent and uniform access to DCE administration functions, wherever they reside, from any and every point in the cell.

DCE daemon (dced).

A continuously running program on each host that provides access to the host services either locally on that host, or remotely from another host.

DCE Directory Service.

The DCE Directory Service is a distributed, replicated database service consisting of a hierarchical set of names which have associated attributes. Given a name, its associated attributes can be looked up in the directory service.

DCE Distributed File Service (DFS).

In DCE, a file service that joins the local file systems of several file server machines, making the file systems equally available to all DFS client machines.

DCE Distributed Time Service (DTS).

A time service that provides fault-tolerant clock synchronization for systems in local area networks and wide area networks. The clock synchronization provided by DTS enables distributed computing applications to determine event sequencing, duration, and scheduling.

DCE Global Directory Service (GDS).

The DCE GDS component is a distributed, replicated directory service based on the CCITT X.500/ISO 9594 international standard. It provides a global namespace that connects the local DCE cells into one worldwide hierarchy.

DCE remote procedure call (RPC).

A call to a procedure in a different address space. In a traditional procedure call, the calling procedure and the called procedure are in the same address space on one machine. In a remote procedure call, the calling procedure invokes a procedure in a different address space, and usually on a different machine. See other glossary terms beginning with binding, interface, and RPC.

DCE Security Service.

The DCE Security Service comprises several parts, including the authentication service, the privilege service, the registry service, the access control list facility, the login facility, and the audit service.

DCE Threads.

A user-level (nonkernel) threads library based on the pthreads interface specified by POSIX in the 1003.4a standard (Draft 4). It consists of an API that gives programmers the ability to create and manipulate threads.

dcecp.

See DCE control program.

dced.

See DCE daemon.

default cell.

Security: With the sec_admin and rgy_edit commands, the cell in which the replica being acted on by the sec_admin command is registered.

default DSA.

The DSA generally used when the user does not specify any particular DSA when connecting to the directory system.

default element.

RPC: An optional profile element that contains a nil interface identifier and object UUID and that specifies a default profile. Each profile can contain only one default element. *See also* default profile, profile, profile element.

default profile.

RPC: A backup profile, referred to by the default element in another profile. The NSI import and lookup operations use the default profile, if present, whenever a search based on the current profile fails to find any useful binding information. *See also* default element, profile.

delegate restrictions.

Restrictions that limit who can act as an intermediary for a particular identity in a call chain.

delegation token.

A checksum over EPAC data, encrypted in the privilege server's key and placed in the A_D field of a PTGT. The token is placed in the A_D field by the privilege server when it enables delegation and when it generates a new delegation chain or impersonated identity.

DES.

See Data Encryption Standard.

descriptor.

1. XOM: The means by which the client and service exchange an attribute value and the integers that denote its representation, type, and syntax.
2. XDS: A defined data structure that is used to represent an OM attribute type and a single value.

descriptor list.

GDS: An ordered sequence of descriptors that is used to represent several OM attribute types and values.

destructor.

A user-supplied routine that is expected to finalize and then deallocate a per-thread context value.

DFS.

See Distributed File Service.

dfsd.

DFS: A program that initializes the cache manager and several daemons on a DFS client machine. It must run each time the client machine reboots for the machine to function as a DFS client.

DIB.

See Directory Information Base.

directory.

1. CDS: A logical unit for storing entries under one name (the directory name) in a CDS namespace. In addition to object entries, a directory can contain soft links and child pointers. You can copy, delete, and control access to a directory. Each physical instance of a directory is called a replica.
2. GDS: A collection of open systems that cooperate to hold a logical database of information about a set of objects in the real world.

Directory Access Protocol (DAP).

GDS: The protocol used by a DUA to access a remote DSA.

directory ID.

See directory identifier.

directory identifier (directory ID).

An identifier for distinguishing several configurations of the directory service within an installation.

Directory Information Base (DIB).

GDS: The complete set of information to which the directory provides access, which includes all of the pieces of information that can be read or manipulated using the operations of the directory. It consists of entries.

Distributed File Service (DFS).

DFS: A file service that joins the local file systems of several file server machines, making the file systems equally available to all DFS client machines.

Directory Information Tree (DIT).

GDS: The DIB considered as a tree, whose vertices (other than the root) are the directory entries.

directory package.

DFS: The part of the DFS cache manager that stores directory (rather than file) caching information.

directory schema.

See schema.

directory service.

GDS: A system using a directory. The directory service consists of the DUA and the directory system. The components of the directory service are connected by a communications network.

directory system.

GDS: A system for managing a directory, consisting of one or more DSAs. Each DSA manages part of the DIB.

Directory System Agent (DSA).

GDS: An Open Systems Interconnection (OSI) application process that is part of the directory.

Directory System Protocol (DSP).

GDS: The protocol by a DSA to access another DSA.

Directory User Agent (DUA).

GDS: An OSI application process that represents a user accessing the directory.

discriminator.

RPC: The data item that determines which union case is currently used.

disk usage.

DFS: A statistic reported by the scout program that indicates space usage on a file server machine's aggregates and partitions. An administrator can use scout to highlight disk usage statistics that exceed specified values. *See also* scout.

dispatcher.

XOM: The software that implements the service interface functions using workspace interface functions.

distinguished encoding.

The restrictions to the Basic Encoding Rules designed to ensure a unique encoding of each ASN.1 value, defined in the X.500 Directory Standards (CCITT X.509).

Distinguished Name (DN).

GDS: One of the names of an object, formed from the sequence of RDNs of its object entry and each of its superior entries.

distinguished value.

GDS: An entry's attribute value that has been designated to appear in the RDN of the entry.

Distributed Computing Environment (DCE).

Services and tools that support the creation, use, and maintenance of distributed applications in a heterogeneous computing environment.

DIT.

See Directory Information Tree.

DN.

See Distinguished Name.

Domain Name Service (DNS).

A hierarchical, distributed naming service which, like the GDS, can act as a higher level connector of DCE cells. *See also* DCE Global Directory Service.

drift.

DTS: The change in a clock's error rate over a specified period of time.

DSA.

See Directory System Agent.

DSP.

See Directory System Protocol.

DTS.

See DCE Distributed Time Service.

DTS entity.

DTS: The server or clerk software on a system.

DUA.

See Directory User Agent.

DUA cache.

GDS: The part of the DUA that stores frequently required information.

dump.

DFS: Generally, the conversion of a fileset's contents into a format suitable for storage on a backup tape and the data object that results from this action. However, the operation need not involve dumping to other media such as tape. *See also* full dump, incremental dump, restore.

dump hierarchy.

DFS: A logical structure in the DFS backup system that defines the parent/child relationship between full and incremental dump levels. *See also* full dump, incremental dump.

dump ID number.

DFS: A unique identification number that the DFS backup system assigns to a dump set. It is distinct from the job ID number assigned to an operation in interactive mode. *See also* job ID number.

dump level.

DFS: An entry in the dump hierarchy recorded in the DFS backup system's backup database. There are two types of dump levels: full and incremental. *See also* full dump, incremental dump.

dump set.

In the DFS backup system, the fileset data that results from dumping a particular fileset family at a given dump level. By implication, all of the data in a dump set was dumped at the same time and in the same manner (fully or incrementally).

dynamic endpoint.

RPC: An endpoint that is generated by the RPC runtime for an RPC server when the server registers its protocol sequences and that expires when the server stops running. *See also* well-known endpoint, endpoint.

effective permissions.

The permissions granted to a principal as a result of a masking operation.

element.

Any of the bits of a bit string, the octets of an octet string, or the octets by means of which the characters of a character string are represented.

encryption key.

A secret value shared between two parties that enables them to communicate securely by using the key to

encrypt and decrypt messages. Some servers store encryption keys in a keytab file. *See also* keytab file, password.

endian.

An attribute of data representation that reflects how certain multioctet data is stored in memory. *See also* big endian, little endian.

endpoint.

RPC: An address of a specific server instance on a host. *See also* dynamic endpoint, well-known endpoint.

endpoint map.

RPC: A system-wide database where local RPC servers register binding information associated with their interface identifiers and object UUIDs. The endpoint map is maintained by the endpoint map service of the RPC daemon. *See also* endpoint map service, RPC daemon.

endpoint map service.

RPC: A service provided by the RPC daemon that maintains a system's endpoint map for local RPC servers. When an RPC client makes a remote procedure call by using a partially bound binding handle, the endpoint map service looks up the endpoint of a compatible local server. *See also* endpoint map, partially bound binding handle, RPC daemon.

entity.

1. CDS: A component of CDS software that you can manage independently of any other component. The CDS control program commands are based on directives targeted for specific entities.
2. DTS: A specific software implementation on a system.

entity type.

DTS: An identifier of an entity that determines its relationship to other components: clerk or server.

entry.

GDS: The part of the DIB that contains information relating to a single directory object. Each entry consists of directory attributes.

entry point vector (EPV).

RPC: A list of addresses for the entry points of a set of remote procedures that implements the operations declared in an interface definition. The addresses are listed in the same order as the corresponding operation declarations.

EPAC.

See extended privilege attribute certificate.

epoch.

A timestamp that identifies directory replicas as being part of the same set. CDS uses the epoch timestamp when it skulks a directory: it finds all replicas of the directory that are in the same epoch and makes their contents consistent. If not all replicas share the same epoch, the skulk aborts. The set directory to new epoch command updates the value of the CDS_Epoch attribute.

epoch number.

DTS: An identifier that a server appends to the time values it sends to other servers. Servers only use time values from other servers with whom they share epoch numbers.

EPV.

See entry point vector.

ERA.

See extended registry attribute.

error.

DTS: The difference between a system's clock value and the computed time.

error tolerance.

DTS: The amount of system clock error to which DCE Distributed Time Service responds by abruptly setting the system clock to the computed time, rather than gradually adjusting the clock.

event class.

Logical grouping of audit events, designated by a name that can be any character string up to 256 characters. Generally, an event class comprises audit events that have some form of commonality.

event class file.

A file that contains the declaration of events that constitute an event class. The name of the event class is the same as the name of the event class file.

event name.

Symbolic name assigned to an audit event, consisting of any character string up to 256 characters. It is used for documentation only, and is not used for any other administrative purpose.

event number.

A 32-bit integer assigned to an audit event. An event number is a tuple made up of a set-id and the event-id. It is used in grouping audit events into event classes.

event-id.

Component of the event number that identifies the audit event.

execution semantics.

RPC: The rules of execution for a remote procedure call, including the effect of multiple invocations on the outcome of a procedure's operation. *See also* at-most-once semantics, broadcast semantics, maybe semantics, idempotent semantics.

expand.

To display the contents of (open) a directory by using the CDS browser. You expand a directory that is closed by double-clicking on its icon. Double-clicking on an expanded directory collapses it.

expiration age.

RPC: The amount of time that a local copy of name service data from an NSI attribute remains unchanged before a request from an RPC application for the attribute requires updating it. *See also* NSI attribute.

explicit binding method.

RPC: The explicit method of managing the binding for a remote procedure call in which a remote procedure call passes a binding handle as its first parameter. The binding handle is initialized in the application code. *See also* automatic binding method, binding handle, implicit binding method.

export.

1. RPC: (1) To place the server binding information associated with an RPC interface or a list of object UUIDs or both into an entry in a name service database. (2) To provide access to an RPC interface.
2. DFS: Offering data or making data available to another system. For example, hosts must export a local DCE LFS or non-LFS aggregate to make it available in the DCE namespace.

extended privilege attribute certificate (EPAC).

Contains authorization information specific to the user, such as groups to which the user belongs. EPACs are used to authorize users; that is, to help a server decide whether users should be granted access to resources that the server manages.

extended registry attribute (ERA).

An attribute attached to a registry object, created using the ERA API interfaces.

fault.

RPC: An exception condition, occurring on a server, that is transmitted to a client.

file exporter.

DFS: The part of a file server machine's kernel that responds to file or directory information requests from the client's cache manager.

file server machine.

DFS: A system that maintains one or more local file systems on disk and makes them available (exports them) to other nodes through the file exporter. *See also* server machine.

file system.

DFS: A mountable subtree of the directory hierarchy.

fileset.

DFS: A hierarchical grouping of files managed as a single unit. DCE LFS supports multiple filesets within a single aggregate; in other file systems used with DFS, filesets are equivalent in size to a partition.

fileset database machine.

DFS: A server machine in a cell that houses the FLDB. *See also* server machine.

fileset family.

DFS: In the DFS backup system, a collection of one or more fileset entries. It defines a group of filesets to be backed up together (at the same time and in the same manner).

fileset family entry.

DFS: A single definition in a DFS backup system fileset family. It defines a collection of filesets in terms of their common site, their prefix, or both. *See also* site.

fileset header.

DFS: Part of the data structure that records information about a fileset. The fileset header records status information such as the the current size of the fileset, the quota of the fileset, and the ID number of the fileset. Information such as the fileset ID is also stored in the entry for the fileset in the FLDB.

fileset ID number.

DFS: A number that uniquely identifies each fileset. The read/write and backup versions of a fileset each have their own fileset ID; all copies of the read-only version share the same fileset ID.

fileset label.

DFS: A file containing information about a fileset, such as its name, fileset ID, unique identifier, type, and status.

fileset location database (FLDB).

DFS: A database that records the location and other status information about available DCE LFS and non-LFS filesets, allowing transparent data access. To be available, a fileset must be exported, registered in the FLDB, and mounted in DFS. The FLDB is maintained by the FL server.

fileset location server (FL server).

DFS: A server process that runs on fileset database machines and maintains the FLDB, which tracks the locations of all DCE LFS and non-LFS filesets.

fileset module.

DFS: The part of the cache manager that maintains a list of accessed filesets, their mounted positions in the global file system tree, and their physical locations.

fileset name.

DFS: A name that uniquely identifies each fileset. All versions of a fileset have the same name; the read-only and backup versions have `.readonly` and `.backup` extensions.

fileset quota.

DFS: A disk space limit that a system administrator imposes on each read/write fileset.

fileset registry.

DFS: The part of the file exporter that stores information about filesets residing on the local machine.

fileset server.

DFS: A server process that runs on all file server machines. It provides the interface for system administrators to perform all tasks that treat a fileset as a unit, including creating, deleting, backing up, cloning, and moving.

filespace.

DFS: The global file system made available to all cells in DCE by DFS. Every entry for a file or directory in DFS resides in the DFS filespace. *See also* `Distributed File Service, DCE`.

filter.

An assertion about the presence or value of certain attributes of an entry in order to limit the scope of a search.

filter directives.

Specifies the conditions that must be satisfied before audit records are written, and where to write these records: the audit trail file or the console.

filter rules.

The prescribed procedure used to resolve overlapping directives from different filters.

filter subject.

Denotes the principal, group, or cell to which the filter applies. The filter subject is the client of the distributed application program that caused the event to occur. A filter is always associated with one and only one filter subject.

first-level DSA.

GDS: A DSA that holds the master entry of a first-level object. *See also* `first-level object`.

first-level object.

GDS: A directory object that is an immediate subordinate to the root.

FL server.

See `fileset location server`.

FLDB.

See `fileset location database`.

flush.

DFS: To force the cache manager to discard data from the local cache, so that the next time an application requests the data, the data must be fetched from the file exporter.

foreign cell.

A cell other than the one to which the local machine belongs. *See also* `local cell`.

foreign cell surrogate.

Principals (whose names are in the form `krbtgt/`*cell_name*) that are maintained in the registry database for the purpose of intercell authentication. To accomplish intercell authentication, the foreign cell surrogates in each cell's registry share a secret key. This secret key is known to both the local and foreign cell's authentication service. It is through their surrogates that two instances of the authentication service are able to convey information about their respective principals to one another, thus enabling a principal from one cell to acquire a ticket to a principal in another cell. *See also* `trust peer`.

full dump.

DFS: A dump set in the DFS backup system that includes all of the data from a fileset. A full dump is different from an incremental dump. *See also* `dump`, `incremental dump`.

full name.

CDS: The complete specification of a CDS name, including all parent directories in the path from the cell root to the entry being named.

full pointer.

RPC: A pointer without the restrictions of a reference pointer.

full restore.

DFS: In the DFS backup system, a full restore returns a fileset to its state when last dumped. The resultant fileset includes data from the last full dump and all subsequent incremental dumps, if any. A full restore is different from a date-specific restore. *See also* `date-specific restore`, `restore`.

fully bound binding handle.

RPC: A server binding handle that contains a complete server address including an endpoint. *See also* partially bound binding handle.

function.

A programming language construct, modeled after the mathematical concept. A function encapsulates some behavior. It is given some arguments as input, performs some processing, and returns some results. Also known as procedures, subprograms or subroutines. *See also* operation.

GDA.

See Global Directory Agent.

GDS.

See DCE Global Directory Service.

generic interface.

The interface, defined at a level that is independent of any particular programming language.

gigabyte (GB).

A unit of measurement for storage capacity equal to 1,073,741,824 (2^{30}) bytes.

Global Directory Agent (GDA).

A DCE component that makes it possible for the local CDS to access names in foreign cells. The GDA provides a connection to foreign cells through either GDS or DNS.

global name.

A name that is universally meaningful and usable from anywhere in the DCE naming environment. The prefix /... indicates that a name is global.

global server.

DTS: A server that frequently provides its clock value to courier servers on other LANs, or infrequently provides its clock value to systems that have failed to obtain the specified number of servers locally.

global set.

DTS: The group of global servers in a network.

glue layer.

DFS: The VFS+ functions that integrate the token and authentication requirements of the DCE environment with the standard VFS functions available to a file system.

group.

1. RPC: A name service entry that corresponds to one or more RPC servers that offer common RPC interface(s), RPC object(s), or both. A group contains the names of the server entries, other groups, or both that are members of the group. *See also* NSI group attribute.
2. Security: Data that associates a named set of principals who can be granted common access rights. Also, the second field of a subject identifier.

group member.

RPC: A name service entry whose name occurs in the group. *See also* group.

group name.

A name that uniquely identifies a group of users to the system.

handle.

RPC: An opaque reference to information. *See also* binding handle, context handle, interface handle, name service handle, thread handle.

high convergence.

A setting that controls the degree to which CDS attempts to keep all replicas of a directory consistent. High convergence means CDS makes one attempt to immediately propagate an update to all replicas. If that attempt fails (for example, if one of the replicas is unavailable), the software schedules a skulk for within 1 hour. Under normal circumstances, a skulk occurs at least once every 12 hours on a directory with high convergence. High convergence is expensive, so constant use of it is not advisable. To control convergence, you modify a directory's CDS_Convergence attribute. *See also* low convergence, medium convergence.

home cell.

See local cell.

host ID.

See network address.

host module.

DFS: The part of the file exporter that associates information with each cache manager's request. This information includes the state of the client that made the call and authentication information about the user who made the request.

idempotent semantics.

RPC: A characteristic of a procedure in which executing it more than once with identical input always produces the same result, without any undesirable side effects; for example, a procedure that reads a particular block of an immutable file is idempotent. DCE RPC supports maybe and broadcast semantics as special forms of idempotent operations. *See also* at-most-once semantics, broadcast semantics, maybe semantics.

IDL.

See Interface Definition Language.

IDL compiler, DCE.

RPC: A compiler that processes an RPC interface definition and optional ACF to generate client and server stubs, header files, and auxiliary files. *See also* Interface Definition Language, stub.

illegal.

A violation of an architecture rule that an implementation is required to report. *See also* unpredictable.

immediate delegation target.

An object on which a client directly performed an operation.

immediate subclass.

A subclass, of a class C, having no superclasses that are themselves subclasses of C.

immediate subobject.

One object that is a value of an attribute of another.

immediate subordinate.

In the DIT, an entry is an immediate subordinate of another if its DN is formed by appending its RDN to the DN of the other entry.

immediate superclass.

The superclass, of a class C, having no subclasses that are themselves superclasses of C.

immediate superior.

In the DIT, an entry is the immediate superior of another if its DN, followed by the RDN of the other, forms the DN of the other entry.

immediate superobject.

One object that contains another among its attribute values.

impersonation.

Transmission of a delegation initiator's identity in a manner than does not preserve the identities of participants in the call chain.

implicit binding method.

RPC: The implicit method of managing the binding for a remote procedure call in which a global variable in the client application holds a binding handle that the client stub passes to the RPC runtime. *See also* automatic binding method, binding handle, explicit binding method.

import.

1. RPC: To obtain binding information from a name service database about a server that offers a given RPC interface by calling the RPC NSI import operation.
2. RPC: To incorporate constant, type, and import declarations from one RPC interface definition into another RPC interface definition by means of the IDL import statement.

inaccessible.

XOM: Said of an object for which the client does not possess a valid designator or handle.

inaccuracy.

DTS: The bounded uncertainty of a clock value as compared to a standard reference.

incremental dump.

DFS: A dump set in the DFS backup system that includes only data from a fileset that changed since the previous dump. An incremental dump is different from a full dump. *See also* dump, full dump.

index priority.

Priority of an attribute type in search queries.

index window.

A navigation aid in the CDS browser. When the namespace is in the display window, dragging the slider up and down the vertical scroll bar produces a rectangular box called the index window. The index window displays the name where the slider is currently positioned; releasing mouse button 1 causes the browser to position that name at the top of the window.

information architecture.

GDS: Describes the representation of the information stored in OM objects and the hierarchical relationships between different classes of OM objects.

initial DSA.

GDS: The master DSA of the directory schema.

initiator.

The initial client in a delegation call chain.

instance.

XOM: An object in the category represented by a class.

instance UUID.

RPC: An object UUID that is associated with a single server instance and is provided to clients to unambiguously identify that instance. *See also* object UUID, server instance.

integrity.

A protection level that can be specified in secure RPC communications that ensures that data transferred between two principals has not been modified in transit.

interface.

See also API, RPC interface, SPI.

interface definition.

RPC: A description of an RPC interface written in the DCE Interface Definition Language (IDL). *See also* RPC interface.

Interface Definition Language (IDL).

RPC: A high-level declarative language that provides the syntax for interface definitions. The file syntax of the IDL interface definition is part of the NCA. *See also* IDL compiler, DCE.

interface handle.

RPC: A reference in code to an interface specification. *See also* interface specification.

interface identifier.

RPC: A string containing the interface's UUID and major and minor version numbers of a given RPC interface. *See also* RPC interface.

interface specification.

RPC: An opaque data structure, generated by the DCE IDL compiler from an interface definition, that contains identifying and descriptive information about an RPC interface. *See also* interface definition, interface handle, RPC interface.

interface UUID.

RPC: The UUID generated for an RPC interface definition via the UUID generator, uuidgen. *See also* interface definition, RPC interface, Universal Unique Identifier (UUID).

intermediary.

A server acting on behalf of an initiator, via delegation or impersonation, making requests to another target server.

intermediate data type.

Any of the basic data types in terms of which the other, substantive data types of the interface are defined.

international character.

A character that is not a member of the DCE PCS character set and so is not guaranteed to be supported in a DCE environment. Programmers writing RPC applications that use international characters build support for them into their applications by using user-provided or DCE RPC features for international character support.

interval.

DTS: The combination of a time value and the inaccuracy associated with it; the range of values represented by a combined time and inaccuracy notation. As an example, the interval 08:00.00I00:05:00 (8 o'clock, plus or minus 5 minutes) contains the time 07:57.00.

invoke ID.

An integer used to distinguish one (directory) operation from all other outstanding ones.

job ID number.

DFS: A number assigned to each operation by the DFS backup system when the backup system is used in interactive mode. It is distinct from the dump ID number assigned to a dump set. *See also* dump ID number.

junction.

A specialized entry in the DCE namespace containing binding information to enable communications between different implementations of the directory service.

Kerberos.

The authentication protocol implemented by DCE shared-secret authentication. Kerberos was developed at the Massachusetts Institute of Technology. In classical mythology, Kerberos was the three-headed dog that guarded the entrance to the underworld.

key.

A value used to encrypt and decrypt data. *See also* encryption key.

key management facility.

A DCE security facility that enables noninteractive principals to manage their secret keys.

keytab file.

A security file that contains the encryption keys for server processes (for example, DFS processes) that run on the host machine. Typically, encryption keys for human principals are not stored in a keytab file. *See also* encryption key.

kilobyte (KB).

A unit of measurement for storage capacity equal to 1024 (2^{10}) bytes.

knowledge reference.

Knowledge that associates, either directly or indirectly, a DIT entry with the DSA in which it is located.

LAN.

See local area network.

leaf entry.

A directory entry that has no subordinates. It can be an alias entry or an object entry.

leap seconds.

An infrequent adjustment to UTC to account for the irregularity of the earth's rotation.

LFS, DCE.

See local file system, DCE.

little endian.

An attribute of data representation that reflects how multioctet data is stored in memory. In little endian representation, the lowest addressed octet of a multioctet data item is the least significant. *See also* big endian.

load balancing.

DFS: Distributing system load evenly across file server machines by placing frequently accessed DCE LFS filesets among available file server machines.

local application thread.

RPC: An application thread that executes within the confines of one address space on a local system and passes control exclusively among local code segments. *See also* application thread, RPC thread, client application thread, server application thread.

local area network (LAN).

A set of computers sharing a network that does not include bridges or WAN links.

local audit trail file.

The audit trail file that is specified by the user, using any convenient pathname. This is specified when the audit daemon is started. If the file name is not an absolute pathname, the file is created in the <dcelocal>/var/audit/adm directory.

local cell.

The cell to which the local machine belongs. *See also* foreign cell.

local DSA.

GDS: A DSA that is resident on the same computer as the DUA.

local file system, DCE (DCE LFS).

DFS: The high-performance, log-based file system provided by DCE. DCE LFS supports multiple filesets within a single aggregate, fileset replication, fast system restarts, and DCE ACLs.

local name.

A name that is meaningful and usable only from within the cell where the entry exists. The local name is a shortened form of a global name. Local names begin with the prefix /.: (or the prefix /: for names in the DFS filespace) and do not contain a cell name.

local server.

DTS: A server that synchronizes with its peers and provides its clock value to other servers and clerks on the same LAN.

local set.

DTS: All of the servers in a particular LAN.

local type.

RPC: A type named in a [represent_as] clause and used by application code to manipulate data that is passed in a remote procedure call as a network type. *See also* network type.

lock token.

DFS: A token that allows a client to place a lock on a range of bytes in a file. Read and write lock tokens are available.

log.

DFS: A record of the actions of a program or system and any changes to data associated with those actions. DCE LFS also maintains a log of changes to metadata on each LFS aggregate.

log-based file system.

DFS: A file system in which changes to metadata are recorded in a log associated with the aggregate on which that file system is located. DCE LFS is a log-based file system. *See also* log.

login facility.

A DCE security facility that enables a principal to establish its identity and assume other identities.

low convergence.

A setting that controls the degree to which CDS attempts to keep all replicas of a directory consistent. Low convergence means CDS does not immediately propagate an update; it simply waits for the next skulk to distribute all updates that occurred since the last skulk. Skulks occur at least once every 24 hours on directories with low convergence. Low convergence helps conserve resources by avoiding update propagations between skulks. To control convergence, you set a directory's CDS_Convergence attribute. *See also* high convergence, medium convergence.

manager.

RPC: A set of remote procedures that implement the operations of an RPC interface and that can be dedicated to a given type of object. *See also* object, RPC interface.

manager entry point vector (manager EPV).

RPC: The runtime code on the server side uses this EPV to dispatch incoming remote procedure calls. *See also* entry point vector, manager.

marshalling.

RPC: The process by which a stub converts local arguments into network data and packages the network data for transmission. *See also* network data, unmarshalling.

mask.

1. With respect to DCE ACLs, a set of permissions that may be intersected (logically ANDed) with another set of permissions associated with a specified privilege attribute in order to yield the effective permissions for principals that possess that privilege attribute.
2. To apply a mask.
3. DFS: A pattern of bits or characters used to control the retention or elimination of portions of another pattern of bits or characters, usually through an AND or OR operation.
4. GDS: Refers to the administration screen interface menus.

mask_obj mask.

When supported by an ACL manager type, the mask_obj mask represents the greatest set of security: an optional alternate name for a primary name in the registry database. Aliases and the primary name for which they are an alternate share the same UUID and UNIX ID permissions granted to principals other than those that match the user_obj or other_obj.

master DSA.

GDS: The DSA that contains the master entry of an object.

master entry.

GDS: The original entry of an object. This is the entry in the DSA that is specified in the master knowledge attribute of the entry.

master information.

GDS: The information from the master entries.

master knowledge attribute.

GDS: An attribute that designates the master DSA of an entry.

master replica.

1. The first instance of a specific directory in the namespace. Once copies of the directory have been made, it is possible to designate a different replica as the master if necessary, but only one master replica of a directory can exist at a time. CDS can create, update, and delete object entries and soft links in a master replica.
2. Security: An instance of a security server that accepts queries and updates to its associated registry database. The master replica dynamically propagates its updates to slave replicas. Each cell has only one master replica. *See also* slave replica.

maybe semantics.

RPC: A form of idempotent semantics that indicates that the caller neither requires nor receives any response or fault indication for an operation, even though there is no guarantee that the operation completed. An operation with maybe semantics is implicitly idempotent and lacks output parameters. *See also* at-most-once semantics, broadcast semantics, idempotent semantics.

medium convergence.

A setting that controls the degree to which CDS attempts to keep all replicas of a directory consistent. Medium convergence means CDS makes one attempt to immediately propagate an update to all replicas of the directory in which a change was just made. If the attempt fails, the software lets the next scheduled skulk take care of making the replicas consistent. Skulks occur at least once every 12 hours on a directory with medium convergence. When you create a namespace, the default setting on the root directory is medium. To control convergence, you set a directory's CDS_Convergence attribute. *See also* high convergence, low convergence.

megabyte (MB).

A unit of measurement for storage capacity equal to 1,048,576 (2^{20}) bytes.

metadata.

The structural data associated with the file system, such as the organization of directories, inode tables, and links. Metadata is not data supplied by a user; it is information about the structure of user data.

minimally consistent.

Said of an object that satisfies various conditions set forth in the definition of its class.

monitoring window.

DFS: A separate terminal session dedicated to tracking the activities of a tape coordinator on a tape coordinator machine. A monitoring window must run on the same machine as the tape coordinator and tape drive it is monitoring.

mount point.

DFS: An access point to a fileset in the DFS file tree. Once a fileset has been mounted, the resulting mount point looks and acts like a directory in the file tree.

mount-level directory.

DFS: The top-level directory of a mounted fileset. It becomes transparently equivalent to the mount point for that fileset after the fileset is mounted. *See also* mount point.

multivalued attribute.

A collection of attribute instances of the same attribute type attached to a single registry object.

mutex.

A synchronization object that provides mutual exclusion among threads. A mutex is often used to ensure that shared variables are always seen by other threads in a consistent state.

name.

GDS: A construct that singles out a particular directory object from all other objects. A name must be unambiguous (that is, denote just one object); however it need not be unique (that is, be the only name that unambiguously denotes the object).

name service handle.

RPC: An opaque reference to the context used by the series of next operations called during a specific NSI search or inquiry.

Name Service Interface (NSI).

RPC: A part of the application programming interface of the RPC runtime. NSI routines access a name service, such as CDS, for RPC applications.

name-based authorization.

Provides a server with the client's principal name. *See also* DCE authorization.

namespace.

A complete set of CDS names (these can include directories, object entries, and soft links) that one or more CDS servers look up, manage, and share. CDS names are stored in directory replicas in clearinghouses at each server. The logical picture of a namespace is a hierarchical tree of all of those directories, with the root directory at the top, and one or more levels of directories beneath the root directory. The physical implementation of the namespace consists of directories replicated in one or more clearinghouses in the network.

naming attribute.

An attribute used to form the RDN of an entry.

NCA.

See Network Computing Architecture.

NDR.

See Network Data Representation.

network address.

RPC: An address that identifies a specific host on a network.

Network Computing Architecture (NCA).

RPC: An architecture for distributing software applications across heterogeneous collections of networks, computers, and programming environments. NCA specifies the DCE RPC architecture.

network data.

RPC: Data represented in a format defined by a transfer syntax. *See also* transfer syntax.

Network Data Representation (NDR).

RPC: The transfer syntax defined by the NEA. *See also* transfer syntax.

network descriptor.

RPC: The identifier of a potential network channel, such as a UNIX socket.

network protocol.

A communications protocol from the Network Layer of the OSI network architecture, such as the IP.

Network Time Protocol (NTP).

Internet-recommended time standard.

network type.

RPC: A type defined in an interface definition and referenced in a [represent_as] clause that is converted into a local type for manipulation by application code. *See also* local type.

NFS/DFS authenticating gateway.

DFS: The NFS/DFS authenticating gateway provides authenticated access to DFS from NFS clients. Users who have DCE accounts can authenticate to DCE via a DFS client configured as a gateway server and access DFS data according to their DCE identities. Administrators can allow users to authenticate to DCE from NFS clients, or administrators can reserve the ability to grant authenticated access from a gateway server only.

node.

A computer connected to a network.

nonspecific subordinate reference.

A knowledge reference that holds information about the DSA that holds one or more unspecified subordinate entries.

NSI.

See Name Service Interface.

NSI attribute.

RPC: An RPC-defined attribute of a name service entry used by the DCE RPC name service interface. An NSI attribute stores one of the following: binding information, object UUIDs, a group, or a profile. *See also* NSI binding attribute, NSI group attribute, NSI object attribute, NSI profile attribute.

NSI binding attribute.

RPC: An RPC-defined attribute (NSI attribute) of a name service entry; the binding attribute stores binding information for one or more interface identifiers offered by an RPC server and identifies the entry as an RPC server entry. *See also* binding information, NSI object attribute, server entry.

NSI group attribute.

RPC: An RPC-defined attribute (NSI attribute) of a name service entry that stores the entry names of the members of an RPC group and identifies the entry as an RPC group. *See also* group.

NSI object attribute.

RPC: An RPC-defined attribute (NSI attribute) of a name service entry that stores the object UUIDs of a set of RPC objects. *See also* object.

NSI profile attribute.

RPC: An RPC-defined attribute (NSI attribute) of a name service entry that stores a collection of RPC profile elements and identifies the entry as an RPC profile. *See also* profile.

NTP.

See Network Time Protocol.

NULL.

The value of a pointer that indicates that the pointer does not point to data.

null binding handle.

RPC: A binding handle containing the NULL value. *See also* binding handle.

object.

1. A data structure that implements some feature and has an associated set of operations.
2. RPC: For RPC applications, an object can be anything that an RPC server defines and identifies to its clients (using an object UUID). Often, an RPC object is a physical computing resource such as a database, directory, device, or processor. Alternatively, an RPC object can be an abstraction that is meaningful to an application, such as a service or the location of a server. *See also* object UUID.
3. XDS: Anything in some ″world,″ generally the world of telecommunications and information processing or some part thereof, that is identifiable (can be named) and for which the DIB contains some information.
4. XOM: Any of the complex information objects created, examined, modified, or destroyed by means of the interface.
5. DFS: A file or directory in a file system. Directories can be further classified as container objects.

object class.

CDS, GDS: An identified family of objects that share certain characteristics. An object class can be specific to one application or shared among a group of applications. An application interprets and uses an entry's class-specific attributes based on the class of the object that the entry describes.

Object Class Table (OCT).

A recurring attribute of the directory schema with the description of the object classes permitted.

object entry.

CDS: The name of a resource (such as a node, disk, or application) and its associated attributes, as stored by CDS. CDS managers, client application users, or the client applications themselves can give a resource an object name. CDS supplies some attribute information (such as a creation timestamp) to become part of the object, and the client application can supply more information for CDS to store as other attributes. *See also* entry.

object identifier.

A value (distinguishable from all other such values) that is associated with an information object. (X.208)

object management.

The creation, examination, modification, and deletion of potentially complex information objects.

object name.

A CDS name for a network resource.

object UUID.

RPC: The universal unique identifier that identifies a particular RPC object. A server specifies a distinct object UUID for each of its RPC objects; to access a particular RPC object, a client uses the object UUID to find the server that offers the object. *See also* object, Universal Unique Identifier.

OCT.

See Object Class Table.

octet.

An 8-bit quantity of data.

OM.

See XOM.

OM attribute.

An OM attribute comprises one or more values of a particular type (and therefore syntax).

OM class.

A static grouping of OM objects, within a specification, based on both their semantics and their form.

opaque.

A piece of data or a data type whose contents are not visible to the application routines that use it.

opaque structure.

A data item or data type whose structure is hidden from the code that is handling it.

Open Systems Interconnection (OSI).

The interconnection of open systems in accordance with ISO standards.

open token.

DFS: A token that grants the right to open a file. The types of tokens available are as follows: normal reading, normal writing, executing, shared reading, and exclusive writing. *See also* token.

operation.

1. A set of step-by-step actions specified by a procedure, function, or routine.
2. RPC: The task performed by a given routine or procedure.
3. GDS: Processing performed within the directory to provide a service, such as a read operation. It is given some arguments as input, performs some processing, and returns some results. An application process invokes an operation by calling an interface function.

organization.

Data that associates a named set of users who can be granted common access rights that are usually associated with administrative policy. Also, the third field of a subject identifier.

orphaned call.

RPC: A call executing in an RPC server after the client that started the call fails or loses communications with the server.

OSI.

See Open Systems Interconnection.

PAC.

See privilege attribute certificate.

package.

A specified group of related OM classes, denoted by an object identifier.

package closure.

The set of classes that need to be supported in order to be able to create all possible instances of all classes defined in the package.

PAG.

See process activation group.

group name.

A binding in which the first parameter is a handle parameter that determines the location of a server of the interface.

parent directory.

Any directory that has one or more levels of directories beneath it in a cell namespace. A directory is the parent of any directory immediately beneath it in the hierarchy.

parent dump level.

DFS: An entry in the dump hierarchy that is used as the reference point for dumps made at an incremental dump level. Both a full dump level and another incremental dump level can serve as a parent. *See also* dump, dump hierarchy, full dump, incremental dump.

parent ID number.

DFS: A fileset ID number stored in a fileset header. If the fileset being examined is a read/write fileset, the parent ID is its fileset ID. If the fileset being examined is a read-only or backup copy of a read/write fileset, the parent ID is the fileset ID of the read/write fileset. *See also* fileset ID number.

password.

A string presented by a principal to prove its identity. The login facility transforms this string to generate an encryption key that is used by the authentication service to authenticate the principal. Server principals usually bypass the string-to-key transformation and present an encryption key to the authentication service for authentication. *See also* encryption key.

PCS.

See Portable Character Set.

peer trust.

A type of trust relationship established between two cells by means of a secret key shared by mutual authentication surrogates maintained by the two cells. A peer trust relationship enables principals in the one cell to communicate securely with principals in the other.

permission.

1. The modes of access to a protected object. In DCE security, the number and meaning of permissions with respect to the object are defined by the ACL manager of the object. *See also* access control list.
2. GDS: One of five groups that assigns modes of access to users: MODIFY PUBLIC, READ STANDARD, MODIFY STANDARD, READ SENSITIVE, or MODIFY SENSITIVE. *See also* access control list.

person.

The name assigned to a DCE principal. The registry database contains the person objects with which accounts can be associated. Also, the first field of a subject identifier.

pickle.

An encoding of a typed value in a byte stream. Pickles are useful for storing or transmitting typed values in typeless media. The type of value contained in a pickle may be understood from context or represented in the pickle itself.

pipe.

1. RPC: A mechanism for passing large amounts of data in a remote procedure call.
2. RPC: The data structure that represents this mechanism.

PKSS.

See private key storage server.

plaintext.

The input to an encryption function or the output of a decryption function. Decryption transforms ciphertext into plaintext.

Portable Character Set (PCS).

The DCE PCS is the group of characters for which DCE guarantees support. The DCE RPC runtime requires that all DCE RPC clients and servers support the DCE PCS. The IDL base type specifiers char and idl_char identify DCE PCS characters.

position (within a string).

The ordinal position of one element of a string relative to another.

position (within an attribute).

The ordinal position of one value relative to another.

potential binding.

RPC: A specific combination of an RPC protocol sequence, RPC protocol major version, network address, endpoint, and transfer syntax that an RPC client can use to establish a binding with an RPC server. *See also* binding, endpoint, network address, RPC protocol sequence, RPC protocol, transfer syntax.

predicate.

1. A Boolean logic term denoting a logical expression that determines the state of some variable(s). For example, a predicate can be an expression stating that ″variable A must have the value 3.″ The control expression used in conjunction with condition variables is based upon a predicate. Use a condition variable to wait for some predicate to become true; for example, to wait for something to be in a queue.
2. Audit Service: The criteria used to select audit records in an audit trail file. This is used in creating audit trail analysis and examination programs that read a select number of records from the audit trail file.

presentation address.

An unambiguous name that is used to identify a set of presentation service access points. Loosely, it is the network address of an OSI service. *See also* address.

Presentation Service Access Point (PSAP).

Address of an OSI communications partner. It addresses an application in a computer.

presented type.

RPC: For data types with the IDL transmit_as attribute, the data type that clients and servers manipulate. Stubs invoke conversion routines to convert the presented type to a transmitted type, which is passed over the network. *See also* transmitted type.

primary alias.

The default name for a cell that has multiple cell aliases. This is the name of the cell that the system will return when asked. *See also* alias.

primary name.

The string name of an object to which any aliases for that object refer. DCE refers to objects by their primary names, although DCE users can refer to them by their aliases.

primary representation.

The form in which the service supplies an attribute value to the client.

primitive binding handle.

RPC: A binding handle whose data type in IDL is handle_t and in application code is rpc_binding_handle_t. *See also* customized binding handle.

principal.

An entity that is capable of believing that it can communicate securely with another entity. In DCE, principals are represented as entries in the registry database and include users, servers, computers, and authentication surrogates.

principal identifier.

The name used to identify a principal uniquely. In DCE, principal identifiers are implemented as UUIDs.

privacy.

A protection level that may be specified in secure RPC communications and that encrypts RPC argument values.

private key.

The key needed by a principal in public key authentication. It is half of the key pair used in public key authentication. The other half is the public key. This method of public and private key pair usage constitutes the public key protocol.

private key storage server.

A server that stores private keys in such a way that only their true owners can retrieve them.

private object.

1. XDS: An OM object created in a workspace by using the object management functions. The term is simply used for contrast with a public object.
2. XOM: An object that is represented in an unspecified fashion.

privilege attribute.

An attribute of a principal that can be associated with a set of permissions. DCE privilege attributes are identity based and include the principal's name, group memberships, and native cell.

privilege attribute certificate (PAC).

Data, describing a principal's privilege attributes, that has been certified by an authority. In DCE, the privilege service is the certifying authority and seals the privilege attribute data in a ticket. The authorization protocol, DCE authorization, determines the permissions granted to principals by comparing the privilege attributes in PACs with entries in an ACL.

privilege required.

DFS: The administrative privilege required to issue a DFS command that affects filesets or DFS server processes. Administrative privilege for a DFS server process is granted to a user who is listed in the administrative list for that server process. *See also* administrative list.

privilege service.

One of the services provided by DCE security; the privilege service certifies a principal's privileges.

procedure declaration.

RPC: The syntax for an operation, including its name, the data type of the value it returns (if any), and the number, order, and data types of its parameters (if any).

process activation group (PAG).

DFS: A unique identifier that the DFS cache manager associates with a user's DCE credentials. The cache manager identifies the user's credentials by the associated PAG to allow the user authenticated access to DFS. Processes forked from the user's login process inherit the PAG to allow for authenticated access to DFS. The cache manager stores the PAG in the kernel of the DFS client.

process entry.

DFS: A definition in the BosConfig file that determines a server process to run, the process's type, and any command parameters used by the process.

profile.

RPC: An entry in a name service database that contains a collection of elements from which NSI search operations construct search paths for the database. Each search path is composed of one or more elements that refer to name service entries corresponding to a given RPC interface and, optionally, a given object. *See also* NSI profile attribute, profile element.

profile element.

RPC: A record in an RPC profile that maps an RPC interface identifier to a profile member (a server entry, group, or profile in a name service database). *See also* group, interface identifier, profile, server entry.

profile member.

RPC: A name service entry whose name occupies the member field of an element of the profile. *See also* profile.

project list.

A list of all the groups in which a principal is a member. The project list is used to determine the principal's access rights to objects. *See also* principal.

protection level.

The degree to which secure network communications are protected.

protocol sequence.

See RPC protocol sequence.

protocol sequence vector.

RPC: A data structure that contains an array-size count and an array of pointers to RPC protocol-sequence strings. *See also* RPC protocol sequence.

PSAP.

See Presentation Service Access Point.

public key.

An authentication protocol that works via public and private key pairs. The protocol is used by security clients and servers to obtain TGTs for users during login, and which is the first part of user-authentication process. This method of public and private key pair usage constitutes the public key protocol.

public object.

1. XOM: An object that is represented by a data structure whose format is part of the service's specification.
2. XDS: A descriptor list that contains all of the OM attributes of an OM object.

purported name.

A construct that is syntactically a name but that has not yet been shown to be a valid name.

RDN.

See Relative Distinguished Name.

read access.

An access right that grants the ability to view CDS data.

read-only fileset.

DFS: A fileset created by replicating a read/write fileset. A read-only fileset is also referred to as a read-only replica or a read-only version *See also* backup fileset, read/write fileset.

read-only replica.

A copy of a CDS directory in which applications cannot make changes. Although applications can look up information (read) from it, they cannot create, modify, or delete entries in a read-only replica. Read-only replicas become consistent with other, modifiable replicas of the same directory during skulks and routine propagation of updates.

read/write fileset.

DFS: The single version of a fileset that houses the modifiable versions of files and directories. The read/write fileset is the original version for which an FLDB entry is allocated. It serves as the source fileset for its associated read-only and backup filesets. It is also referred to as the read/write source or read/write version. *See also* backup fileset, read-only fileset.

read/write mount point.

DFS: A type of mount point that instructs the cache manager to access only the exact fileset specified in the mount point, not its read-only version. *See also* mount point, regular mount point.

realm.

A cell, considered exclusively from the point of view of security; this term is used in Kerberos specifications. In DCE documentation, the term ″cell'' designates the basic unit of DCE configuration and administration, and incorporates the notion of a realm.

recurring attribute.

An attribute with several attribute values.

redirection.

The act of changing the standard use of input and output to a user-specific method. For example, standard output can be redirected to a file.

reentrant service.

A service that is safe to call from multiple threads in parallel. If a service is reentrant, there is no burden placed on calling routines to serialize their access or take other explicit precautions. *See also* thread-serial service, thread-synchronous service.

reference monitor.

Code that controls access to an object. In DCE, servers control access to the objects they maintain; and for a given object, the ACL manager associated with that object makes authorization decisions concerning the object.

reference pointer.

RPC: A non-null pointer whose value is invariant during a remote procedure call and cannot point at aliased storage.

referral.

An outcome that can be returned by a DSA that cannot perform an operation itself. The referral identifies one or more other DSAs more able to perform the operation.

register.

1. RPC: To list an RPC interface with the RPC runtime.
2. RPC: To place server-addressing information into the endpoint map.
3. RPC: To insert authorization and authentication information into binding information. *See also* endpoint map, RPC interface.

registry database.

A database of information about persons, groups, organizations, and accounts.

registry object.

A data node in the registry database. Registry objects are of the following object types: principal, group, org, directory, policy, replist (replica list), and xattrschema. There are many nodes of the principal, group, org and directory types. There is only one node each for the policy, replist and xattrschema types.

registry replica.

A read-only instance of a registry database.

registry service.

One of three services provided by DCE security; the registry service manages account information for principals. The other services are the privilege service and the authentication service.

regular mount point.

DFS: The most common type of mount point. If the fileset it names is a read/write fileset, the cache manager is free to access a read-only version of the fileset (if one exists). *See also* mount point, read/write mount point.

Relative Distinguished Name (RDN).

A set of Attribute Value Assertions (AVAs), each of which is true, concerning the distinguished values of a particular entry.

relative time.

A discrete time interval that is usually added to or subtracted from an absolute time.

release replication.

DFS: A method of updating read-only copies of filesets. Release replication is not automatic like scheduled replication; each update must be initiated by an administrator. *See also* replication, scheduled replication.

remote procedure.

RPC: An application procedure located in a separate address space from the calling code. *See also* remote procedure call.

remote procedure call (RPC).

RPC: A procedure call executed by an application procedure located in a separate address space from the calling code. *See also* remote procedure.

replica.

1. CDS: a copy of a directory in the CDS namespace. The first instance of a directory in the namespace is the master replica. When CDS managers make copies of the master replica to store in other clearinghouses, all of the copies, including the master replica, become part of the directory's replica set. *See also* read-only replica.
2. DFS: A read-only copy of a fileset that contains all the data of the source fileset. As a full copy of a fileset, a replica can exist on any aggregate. A replica is different from a clone, which can reside only on the same aggregate as the source fileset. *See also* clone.
3. Security: An instance of the security server and its database. One replica, the master replica, can accept updates and queries to its database. The slave replica can accept only queries.

replica set.

The set of all copies of a CDS directory. Information about a directory's replica set is contained in an attribute of directories and child pointers called CDS_Replicas. The attribute contains the type of each replica (master or read-only) and the clearinghouse where it is located. When skulking a directory, CDS refers to the directory's replica set to ensure that it finds all copies of that directory. During a lookup, CDS can refer to the replica set in a child pointer when trying to locate a directory that does not exist in the local clearinghouse.

replication.

1. CDS: Making a copy of a CDS directory in another clearinghouse. Replication can improve availability and load sharing. *See also* replica.
2. GDS: The process by which copies of objects are created and maintained.
3. DFS: The process of creating read-only copies of a fileset. In DFS, there are two types of replication: release replication and scheduled replication. Replication is supported only for DCE LFS filesets. *See also* release replication, scheduled replication.

replication server.

DFS: A server process used in release replication and scheduled replication. The replication server tracks the currency of read-only replicas of filesets. It updates each replica to match its read/write source fileset as appropriate. *See also* replication.

request buffer.

RPC: A first-in, first-out queue where an RPC system temporarily stores call requests that arrive at an endpoint of an RPC server, until the server can process them.

restore.

DFS: The translation of a previously dumped fileset back into fileset format and its eventual replacement in the file system. The DFS Backup System allows several different types of restores, including full restores and date-specific restores. The operation need not involve recovery from other media such as tapes. *See also* date-specific restore, dump, full restore.

return value.

A function result that is returned in addition to the values of any output or input/output arguments.

RPC.

See also remote procedure call, DCE remote procedure call.

RPC control program.

RPC: An interactive management facility for managing name service entries and endpoint maps for RPC applications. The program is started by the rpccp command.

RPC interface.

RPC: A logical grouping of operation, data type, and constant declarations that serves as a network contract for calling a set of remote procedures. *See also* interface definition.

RPC protocol.

RPC: An RPC-specific communications protocol that supports the semantics of the DCE RPC API and runs over either connectionless or connection-oriented communications protocols.

RPC protocol sequence.

RPC: A valid combination of communications protocols represented by a character string. Each protocol sequence typically includes three protocols: a network protocol, a transport protocol, and an RPC protocol that works with those network and transport protocols. *See also* network protocol, RPC protocol, transport protocol.

RPC runtime.

RPC: A set of operations that manages communications, provides access to the name service database, and performs other tasks, such as managing servers and accessing security information, for RPC applications. *See also* RPC runtime library.

RPC runtime library.

RPC: Routines of the RPC runtime that support the RPC applications on a system. The runtime library provides a public interface to application programmers, the application programming interface (API), and a private interface to stubs, the stub programming interface (SPI). *See also* RPC runtime.

RPC thread.

RPC: A logical thread within which a remote procedure call executes. *See also* thread.

rundown procedure.

RPC: A procedure, typically used with a context handle, that is called following a communications failure to recover resources reserved by a server for servicing requests by a particular client. *See also* context handle.

S-stub.

The part of the DSA that establishes the connection to the communications network.

salvager.

DFS: A program that finds and attempts to repair inconsistencies in DCE LFS aggregates. The salvager is similar to the fsck program in other, non-LFS file systems.

scheduled replication.

DFS: A method of updating read-only copies of filesets. Scheduled replication is automatically performed by the replication server at specified intervals. *See also* release replication, replication.

schema.

The directory schema is the set of rules and constraints concerning the DIT structure, object class definitions, attribute types, and syntaxes that characterize the DIB. *See also* attribute schema.

schema entry.

A record containing the identifiers and characteristics of an attribute type. A schema entry is essentially an attribute type definition.

schema object.

The registry data node, with the well-known name xattrschema (under the security junction point, typically /.:/sec), containing the attribute schema information. (Also called attribute schema object.)

scout.

DFS: A program that can be run on any machine configured as a DFS client. It monitors the file exporter running on designated file server machines by periodically collecting statistics and displaying them in a graphical format. *See also* attention threshold, basename, disk usage.

seal.

To encrypt a record containing several fields in such a way that the fields cannot be modified without either knowledge of the encryption key or leaving evidence of tampering.

secondary representation.

A second form, an alternative to the primary representation, in which the client can supply an attribute value to the service.

secondary site.

DFS: A read-only site that receives updates to its copy of a DFS administrative database from the Ubik synchronization site. There can be more than one secondary site. If necessary, a secondary site can be elected to assume the role of synchronization site. *See also* synchronization site, Ubik.

secret key.

A long-lived encryption key known to more than one principal, usually two. In DCE, each secret key is known to the Authentication Service and one other principal.

security. *See* DCE Security Service.

segment.

Zero or more contiguous elements of a string.

self-pointing type.

RPC: A data type containing a pointer member that can point directly or indirectly to another item of the same type.

SEP line.

In an event class file, an entry that specifies the prefixes of the event numbers in the file. This is an optional entry and is used to speed up the search for events in event class files.

server.

1. RPC: The party that receives remote procedure calls. A given application can act as both an RPC server and an RPC client. *See also* client.
2. CDS: A node running CDS server software. A CDS server handles name-lookup requests and maintains the contents of the clearinghouse or clearinghouses at its node.
3. DTS: A system or process that synchronizes with its peers and provides its clock value to clerks and their client applications.
4. DFS: A provider of resources or services. *See also* client.
5. GDS: The server consists of a DSA, which accesses the database, and an S-stub, which handles the connection over the communications network for responding to remote clients and accessing remote servers.

server addressing information.

RPC: An RPC protocol sequence, network address, and endpoint that represent one way to access an RPC server over a network; a part of server binding information. *See also* binding information, endpoint, network address, RPC protocol sequence.

server application thread.

RPC: A thread executing the server application code that initializes the server and listens for incoming calls. *See also* application thread, client application thread, local application thread, RPC thread.

server binding information.

RPC: Binding information for a particular RPC server. *See also* binding information, client binding information.

server entry.

1. RPC: A name service entry that stores the binding information associated with the RPC interfaces of a particular RPC server and also the object UUIDs for any objects offered by the server. *See also* binding information, NSI binding attribute, object, NSI object attribute, RPC interface.
2. DFS: A unique identifier for a server machine in the FLDB.

server instance.

RPC: A server executing in a specific address space; multiple server instances can coexist on a single system. *See also* server.

server machine.

DFS: A machine that runs one or more DFS server processes. Depending on the process it runs, a server machine can be further classified as a file server machine, a system control machine, a binary distribution machine, a fileset database machine, or a backup database machine. *See also* client machine.

server module.

DFS: The part of the DFS cache manager that provides information for tracking server activity.

server portion of update server.

See upserver.

server process.

DFS: A process that runs on server machines, providing services such as storing and transferring files or tracking fileset locations to clients. *See also* server machine.

server stub.

RPC: The surrogate calling code for an RPC interface that is linked with server application code containing one or more sets of remote procedures (managers) that implement the interface. *See also* client stub, manager, stub.

service.

RPC: An integral set of RPC interfaces offered together by a server to meet a specific goal. *See also* RPC interface.

service controls.

A group of parameters, applied to all directory operations, that direct or constrain the provision of the service.

session.

A sequence of directory operations requested by a particular user of a particular DUA. The operations use the same session OM object.

session key.

Used in Kerberos specifications; acronym for ''conversation key.'' *See also* conversation key.

set-id.

In event numbers, the component of the event number that identifies a set of events to which the audit event belongs. In event class numbers, the component of the event class number that identifies a set of event classes to which the event class belongs.

shadow entry.

A copy entry of an object. This is an entry of an object in a DSA other than the master DSA.

signal.

Threads: To wake only one thread waiting on a condition variable. *See also* broadcast.

signed.

Information is digitally signed by appending to it an enciphered summary of the information. This is used to

ensure the integrity of the data, the authenticity of the originator, and the unambiguous relationship between the originator and the data.

simple bnode.

DFS: A bnode that manages a single process that is to be kept running at all times. *See also* bnode, basic overseer server.

simple name.

One element in a CDS full name. Simple names are separated by / (slashes).

simple object.

An object that does not hold other objects. For example, a file is a simple object. *See also* container object.

simple process.

DFS: A type of process defined in a server machine's BosConfig file. It runs continuously and can be stopped and restarted independently of any other process on its machine. *See also* cron process, simple bnode.

site.

DFS: The location of a fileset expressed as a specific file server machine and aggregate.

site count.

DFS: A count of the number of sites where the read/write and read-only versions of a fileset reside.

site flags.

DFS: A term for the flags associated with each site definition in an FLDB entry. The flags can indicate the fileset type (read/write or read-only) and other administrative information.

skew.

The time difference between two clocks or clock values.

skulk.

A process by which CDS makes the data consistent in all replicas of a particular directory. CDS collects all changes made to the master replica since the last skulk completed, and disseminates the changes from the up-to-date replica to all other existing replicas of the directory. All replicas of a directory must be available for a skulk to be considered successful. If a skulk fails, CDS informs you of the replicas that it could not reach.

slave replica.

An instance of a security server that accepts only queries to its associated registry database. Slave replicas are updated by the master replica. Each cell can have many slave replicas. *See also* master replica.

soft link.

A pointer that provides an alternate name for an object entry, directory, or other soft link in the namespace. A soft link can be permanent or it can expire after a period of time that you specify. The CDS server also can delete it automatically after the name that the link points to is deleted.

source fileset.

See read/write fileset.

specific.

The attribute types that can appear in an instance of a given class, but not in an instance of its superclasses.

SPI.

See stub programming interface.

SRT.

See Structure Rule Table.

status flag.

DFS: In a BosConfig file, the flag that tells the BOS server whether a server process should be running. In an FLDB entry, the flag that indicates whether a fileset of each possible type (read/write, read-only, and backup) actually exists at a site. In a fileset header, a flag that indicates whether the contents of the fileset are accessible via the file server machine.

status token.

DFS: A token that grants access to the status information associated with a file or directory. Read and write status tokens are available.

string.

An ordered sequence of bits, octets, or characters, accompanied by the string's length.

Structure Rule Table (SRT).

A recurring attribute of the directory schema with the description of the permitted structures of distinguished names.

stub.

RPC: A code module specific to an RPC interface that is generated by the DCE IDL compiler to support remote procedure calls for the interface. RPC stubs are linked with client and server application and hide the intricacies of remote procedure calls from the application code. *See also* client stub, server stub.

stub programming interface (SPI).

A private RPC runtime interface whose routines are unavailable to application code.

subclass.

One of the classes, designated as such, whose attribute types are a superset of those of another class.

subobject.

An object that is in a subordinate relationship to a given object.

subordinate.

In the DIT, an entry is subordinate to another if its distinguished name includes that of the other as a prefix.

superclass.

One of the classes, designated as such, whose attribute types are a subset of those of another class.

superior.

In the DIT, an entry is superior to another if its distinguished name is included as a prefix of the distinguished name of the other. Each entry has exactly one immediate superior.

superobject.

An object that is in a superior relationship to a given object.

synchronization.

DTS: The process by which a DTS entity requests clock values from other systems, computes a new time from the values, and adjusts its system clock to the new time.

synchronization list.

DTS: The list of servers that a DTS entity has discovered; the entity sends requests for clock values to the servers on the list.

synchronization site.

DFS: The one Ubik site that accepts changes to its copy of a DFS administrative database and distributes them to the secondary sites. The synchronization site can change as necessary. *See also* `secondary site`, `Ubik`.

syntax.

XOM: (1) An OM syntax is any of various categories into which the object management specification statically groups values on the basis of their form. These categories are additional to the OM type of the value. (2) A category into which an attribute value is placed on the basis of its form. *See also* `attribute syntax`.

syntax template.

A lexical construct containing an asterisk from which several attribute syntaxes can be derived by substituting text for the asterisk.

system control machine.

DFS: The machine that distributes common configuration files to other server machines in the cell or administrative domain. The system control machine runs the server portion of the update server for this purpose. *See also* `server machine`, `update server`, `upserver`.

system time.

The time value that the operating system maintains according to its reading of the system's hardware clock.

tape coordinator.

DFS: A process that runs on a tape coordinator machine and controls the behavior of one tape drive. There must be one tape coordinator running for each tape drive in use.

tape coordinator ID (TCID).

DFS: A number, assigned when a tape coordinator machine is configured, that uniquely identifies each tape coordinator and the associated tape drive. Backup operators use it to specify the tape coordinator that is to execute a command.

tape coordinator machine.

DFS: A client machine on which backup and restore operations are performed with the DFS backup system. Each tape coordinator machine must have one tape drive attached and must run one instance of the `butc` process for each drive.

target restrictions.

Restrictions on the targets to whom a client's delegated identity can be projected.

TCID.

See `tape coordinator ID`.

TCP.

See `Transmission Control Protocol`.

TDF.

See `Time Differential Factor`.

thread.

A single sequential flow of control within a process.

thread handle.

RPC: A data item that enables threads to share a memory management environment.

thread-serial service.

A reentrant system service is thread-serial if it blocks the current thread and all other threads that attempt to call the same service or other related services until the first call returns.

thread-synchronous service.

A reentrant system service is thread-synchronous if it blocks only the current thread and allows other threads to execute the same operation during the block.

tick.

DTS: The clock timer interrupt that causes the operating system to increment the system time.

ticket.

An application-transparent mechanism that transmits the identity of an initiating principal to its target. A simple ticket contains the principal's identity, a session key, a timestamp, and other information, sealed using the target's secret key. A privilege ticket contains the same information as a simple ticket, and also includes a privilege attribute certificate. A ticket-granting ticket is ticket to the ticket-granting service; a service ticket is a ticket for a specified service other than the ticket-granting service.

Time Differential Factor (TDF).

DTS: The difference between UTC and the time in a particular time zone.

time-provider.

DTS: A hardware device that monitors UTC time and forwards it to a DTS server.

Time-Provider Interface (TPI).

A software intermediary between the DTS server and external time-provider processes. The DTS server uses the interface to obtain UTC time values and to determine the associated inaccuracy of each value.

time-provider program.

DTS: Software that enables a time-provider device to call the time-provider interface and supply time values to a DTS server.

timeslicing.

A mechanism by which running threads are preempted at fixed intervals. This ensures that every thread is allowed time to execute.

token.

DFS: A device sent along with requested data from a file server machine to a client machine to indicate the types of operations (for example, read or write) the client can perform on the data. It prevents simultaneous access while permitting cooperative access; for example, only one client can possess a write token for a single piece of data at any given time. A client must have the appropriate tokens to operate on a file exporter. *See also* data token.

token management layer.

DFS: The part of the DFS cache manager that handles file and directory tokens. *See also* token manager.

token manager.

DFS: A component that maintains the set of file and directory tokens that have been granted to existing clients of a file server machine. *See also* token management layer.

top-level pointer.

RPC: A pointer parameter that, in a chain of pointers, is the only member that is not the referent of any other pointer.

tower.

Physical address and protocol information for a particular server. CDS uses this information to locate the system on which a server resides and to determine which protocols are available at the server. Tower values are contained in the CDS_Towers attribute associated with the object entry that represents the server in the cell namespace.

TP server.

DTS: A server system connected to a time-provider.

TPI.

See Time-Provider Interface.

traced delegation.

Transmission of a delegation initiator's identity in a manner that preserves the identities of each participant in a call chain.

transaction.

A related set or unit of changes to metadata. The events in a transaction are atomic. No change takes effect unless all the changes that make up that transaction are performed. *See also* log.

transfer syntax.

RPC: A set of encoding rules used for transmitting data over a network and for converting application data to and from different local data representations. *See also* Network Data Representation.

transmissible.

Able to send a data type over the network by way of an RPC operation.

Transmission Control Protocol (TCP).

A protocol of the Internet Protocol (IP) family.

transmitted type.

RPC: For data types with the IDL transmit_as attribute, the data type that stubs pass over the network. Stubs invoke conversion routines to convert the transmitted type to a presented type, which is manipulated by clients and servers. *See also* presented type.

transparent access.

DFS: A feature that allows users to access files without needing to know which machine stores the files. The FLDB keeps track of fileset locations, so the user needs to know only a file's pathname. *See also* fileset location database.

transport independence.

RPC: The capability, without changing application code, to use any transport protocol that both the client and server systems support, while guaranteeing the same call semantics. *See also* transport layer, transport protocol.

transport layer.

A network service that provides end-to-end communications between two parties, while hiding the details of the communications network. The TCP and ISO TP4 transport protocols provide full-duplex virtual circuits on which delivery is reliable, error free, sequenced, and duplicate free. UDP provides no guarantees (the connectionless RPC protocol provides some guarantees on top of UDP).

transport protocol.

A communications protocol from the transport layer of the OSI network architecture, such as the TCP or the UDP.

trigger.

A remote operation, associated with an attribute type, that is executed when attributes of that type are either queried or updated.

trigger type.

A classification, either query or update, on a trigger that identifies on which attribute operation the trigger will be invoked.

trust path.

See authentication path.

trust peer.

A characterization of one cell with respect to another with which the cell maintains a mutual authentication surrogate.

type.

XOM: A category into which attribute values are placed on the basis of their purpose. *See also* attribute type.

type UUID.

RPC: The universal unique identifier that identifies a particular type of object and an associated manager. *See also* manager, object, Universal Unique Identifier.

Ubik.

DFS: A library of utilities that the DFS FL server and the DFS backup server use to keep individual copies of the FLDB and the backup database. *See also* secondary site, synchronization site.

UFS.

See UNIX File System.

unauthenticated mask.

The maximum set of permissions granted when access to the object is not authenticated. Unauthenticated access is granted only if the caller matches some ACL entry (frequently the any_other entry) and if the required permission is granted by both the entry and the mask.

unconfigure.

To remove configuration data.

unexport.

RPC: To remove binding information from a server entry in a name service database. *See also* export.

uniquifier.

DFS: A piece of data that, in combination with a fileset ID, produces a globally unique identifier.

Universal Unique Identifier (UUID).

RPC: An identifier that is immutable and unique across time and space. A UUID can uniquely identify an entity such as an RPC interface or object. *See also* interface UUID, object UUID, type UUID.

UNIX File System (UFS).

A section of the UNIX file tree that is physically contained on a single device or disk partition and that can be separately mounted, dismounted, and administered.

unmarshalling.

RPC: The process by which a stub disassembles incoming network data and converts it into local data in the appropriate local data representation. *See also* marshalling, network data.

unpickle.

A decoding of a typed value in a byte stream.*See also* pickle.

unpredictable.

A violation of an architecture rule that an implementation is not required to report. Results can include an error report from a threads call, the operating system, or the hardware; a hang or deadlock of the program; or an incorrect operation of the program without indication of error.

unreachable.

At your current location in the network, you cannot get to your destination. For example, the network address may be in use or a physical network connection may not exist or be operational.

upclient.

DFS: A process that runs on DFS server machines, taking copies of common configuration files and new DFS server process binary files from central sources. *See also* update server, upserver.

update propagation.

An immediate attempt to apply a change to all replicas of the CDS directory in which the change was just made. An update propagation delivers changes in a more efficient and timely way than a skulk, which is the periodic distribution of a whole collection of changes.

update server.

DFS: A process that guarantees that all DFS server machines in a cell have the same versions of common configuration files and the same versions of DFS binary files appropriate for their machine types. It has a server portion called the upserver and a client portion called the upclient. *See also* upclient, upserver.

Update Timestamp (UTS).

An attribute that identifies the time at which the most recent change was made to any attribute of a particular CDS name. For directories, the UTS reflects changes made only to attributes that apply to the directory as a whole (not one of its replicas).

upserver.

DFS: A process that runs on DFS server machines, making local copies of common configuration files and new DFS server process binary files available to other DFS server machines. *See also* upclient, update server.

user.

GDS: The end user of the directory; the entity or person that accesses the directory. A user can be an application program that is calling the directory interface on behalf of a human user.

user data.

DFS: Data such as ASCII and binary files that resides in a fileset. The data is manipulated and accessed by users of the file system.

User Datagram Protocol (UDP).

A protocol of the Internet Protocol (IP) family.

user-to-user.

A protocol used in the ticket-granting process of security authentication of an application.

UTC.

See Coordinated Universal Time.

UTS.

See Update Timestamp.

UUID.

See Universal Unique Identifier.

V file.

DFS: With disk caches, a file on the disk that, by default, can hold up to 64 kilobytes of cached data. A maximum of 32,000 V files can be used for one disk cache.

value.

XOM: An arbitrarily complex information item that can be viewed as a characteristic or property of an object. *See also* attribute value.

varying array.

RPC: An array whose elements do not all need to be transmitted during a remote procedure call.

vector.

RPC: An array of other structures and the number of array items.

VFS.

See Virtual File System.

VFS+.

DFS: Extensions to the standard UNIX Virtual File System (VFS). *See also* Virtual File System.

Virtual File System (VFS).

DFS: A level of abstraction above the specific interfaces to various types of file systems. It is used to avoid having to change kernel code to handle low-level, system-specific differences.

vnode.

DFS: The structure used to access the inode or anode structure associated with a specific file through a virtual file system interface. The term vnode stands for virtual node. *See also* anode.

WAN.

See wide area network.

WAN links.

Communications connections between groups of computers that are spread across a large geographical distance. Modem connections, T1 lines, and satellite hookups are some common examples. *See also* wide area network.

well-known endpoint.

RPC: A preassigned, stable endpoint that a server can use every time it runs. Well-known endpoints typically

are assigned by a central authority responsible for a transport protocol. An application declares a well-known endpoint either as an attribute in an RPC interface header or as a variable in the server application code. *See also* `dynamic endpoint`, `endpoint`.

wide area network (WAN).

A network that includes computers spread across a large geographical distance, usually involving several cities, states, or countries.

workspace.

XDS: A space in which OM objects of certain OM classes can be created, together with an implementation of the object management functions that supports those OM classes.

workspace interface.

The interface as realized, for the dispatcher's benefit, by each workspace individually.

X.500. A set of standards covering electronic directory services. Namespace identifications such as `/.../C=US/O=IBM/OU=DCE` (where c=country, o=organization, and ou=organizational unit) follow the X.500 standard.

XDS.

X/Open Common Application Environment specification for Directory Services API.

XOM.

X/Open Common Application Environment specification for OSI-Abstract-Data Manipulation API.

Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created

programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department LZKS
11400 Burnet Road
Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written.

These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _*enter the year or years.*_ All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

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

- AIX
- IBM

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

DFS is a trademark of the Transarc Corporation.

Other company, product, and service names may be trademarks or service marks of others.

Index

M

N

O

P

R

S

T

U

X

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.