OSF™DCE

Introduction to OSF™DCE

OPEN SOFTWARE FOUNDATION

Introduction to OSF™ DCE

Revision 1.0

Open Software Foundation

P T R Prentice Hall, Englewood Cliffs, New Jersey 07632

Cover design: BETH FAGAN
Cover illustration: STEVE LEWONTIN

This book was formatted with troff.

Published by P T R Prentice-Hall, Inc.
A Simon & Schuster Company
Englewood Cliffs, New Jersey 07632

Printed in the United States of America
10 9 8 7 6 5 4 3 2

ISBN 0-13-490624-1

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Contents

List of Figures

List of Tables

Preface

Introduction to OSF DCE provides an introduction to the OSF™ 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 is written for anyone interested in an overview of DCE, including managers, system administrators, and application programmers. Chapter 2 is intended for network managers and administrators. Chapters 3 and 4 are targeted primarily for administrators and programmers.

Appendix A 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 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 *OSF DCE Administration Guide* is targeted for the same audience as the *OSF DCE Administration Guide* itself.

Applicability

This is Revision 1.0 of this manual. It applies to the OSF DCE 1.0 offering.

Purpose

After reading this document, a user will

- Have a high-level understanding of DCE
- Understand the individual technology components comprising DCE
- Understand the interdependencies of the DCE technology components
- Be able to find further information about DCE in related documents

Document Usage

The manual is organized as follows:

- Chapter 1

  Chapter 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 comprise DCE, and their integration with one another.

- Chapter 2

  Chapter 2 gives examples of typical DCE configurations. It explains the concept of a DCE cell, and describes the DCE software configuration components. It describes the configuration of different types of DCE machines. It then gives examples of different cell configurations, including a simple DCE cell, and cells with various combinations of DCE services.

- Chapter 3

  Chapter 3 describes each of the technology components that comprise DCE. It includes sections on DCE Threads, Remote Procedure Call, Directory Service, Distributed Time Service, Security Service, Distributed File Service, and Diskless Support Service. Its last section shows how some of these services are used in a simple distributed application example.

- Chapter 4

  The DCE technologies are integrated with one another; that is, they use each other's services wherever appropriate. Chapter 4 describes the ways in which each of the DCE components uses the other technology components of DCE, and what implications their integration has for porting, testing, configuring, and starting up DCE systems.

- Appendix A

  Appendix A gives an overview of DCE documentation, and suggests reading paths for different audiences.

- Appendix B

  Appendix B gives an outline of the contents of the DCE documentation set.

- Appendix C

  Appendix C lists the acronyms and abbreviations used in this manual.

- Glossary

  The glossary defines terms used in this manual and the rest of the DCE documentation set.

Related Documents

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

- *Introduction to OSF DCE*
- *OSF DCE User's Guide and Reference*
- *OSF DCE Application Development Guide*
- *OSF DCE Application Development Reference*
- *OSF DCE Administration Guide*
- *OSF DCE Administration Reference*
- *OSF DCE Porting and Testing Guide*
- *Application Environment Specification/Distributed Computing*
- *OSF DCE Technical Supplement*
- *OSF DCE Release Notes*

Acknowledgements

DCE Revision 1.0 is the result of the work of Gary Aberle, Robert Acosta, Teresa Acosta, R.K. Aditham, Maria Aicher, Mark Akers, Pervaze Akhtar, Vijay Anand, Ted Anderson, Dick Annicchiarico, Vasilis Apostolides, Larry Augustus, Brian Bailey, Anju Bansal, Jeff Barry, Ko Baryiames, Steve Bertrand, Andrew Birrell, Michael Blackstock, Sumner Blount, Winfrid Blumann, Gil Bogo, Beth Bottos, John Bowe, Mary Brady, Michael Brady, Gertjan van den Brandhof, Bill Brown, Josef Buchenberg, Terri Buchman, Mike Burati, Carl Burnett, Lisa Burns, Ann Burton, Bob Burton, Yakov Burtov, Sandra Bushnell, Dave Butenhof, Marysia Cahill, Gerald Cantor, Martine Carannante, Greg Carpenter, Noreen Casey, Debbie Caswell, Monica Cellio, Alex Chen, David Cheriton, David Chinn, Sailesh Chutani, Danny Cobb, Mike Comer, Joe Comuzzi, Elizabeth Connolly, Orla Connolly, Bob Conti, Eileen Coons, Cesar Cortes, Laurie Cura, John Curry, Paul Curtin, Richard Curtis, Sue-fen Cuti, Beth Cyr, Ron Czik, Fred Dalrymple, Sachin Danait, Martha DasSarma, Manfred Demus, Rajendra

Desai, Marcia Desmond, Terry Dineen, Chris Doherty, Nick Dokos, Emer Donnelly, Amy Dorman, Eddie Doyle, Niamh Doyle, John Dugas, Teodor Dumitrescu, Philip Dunne, Donna Esterling, Craig Everhart, Rick Fadden, Patrick Fantou, Dietmar Fauth, Gary Fernandez, Alan Finger, Tony Fiore, Dick Flowers, Ken Flowers, Roger Fondrillon, Mark Fox, Carolyn Francisco, Tonie Franz, Jean Fullerton, Mary Gacoin, Prasad Ganni, Dave Geise, Frank Ginac, Ken Goach, Josh Goldman, Bob Goldschneider, Jonathan Gossels, Roger Gourd, John Grober, Paul Groff, Michael Gross, Praveen Gupta, Bill von Hagen, Alan Hamilton, Debbie Hamilton, Eric Hamilton, Bill Hankard, Peter Harbo, Ward Harold, Melanie Harper, Jerry Harrow, Doug Hartman, Bob Hathaway, Sally Hehir, Walter Heldmann, Mark Heroux, Juergen Herrmann, Ann Hewitt, Mark Hickey, Bill Hileman, Susanna Hill, Liz Hines, Tony Hinxman, Phil Hirsch, Ken Hobday, Heinz Hoffman, Angie Holloway, Tony Hooten, Anne Hopkins, Reinhard Horn, Grace Hsiao, Wei Hu, Liz Hughes, Christian Huitema, Peter Hurley, Jim Jackson, Reinhard Jahn, Sanjay Jain, Vicky Janicki, Eric Jendrock, Brad Johnson, Kathleen Johnson, Melissa Johnston, Ed Jones, Jeff Kaminski, John Kaminsky, Sandhya Kapoor, Mark Karuzis, Don Kassebaum, Greg Kattawar, Marsha Kaufman, Mike Kazar, Katie Kean, Brian Keane, Peter Keegan, David Kenney, Andy Klein, Natasha Kogan, Mike Kong, Kothari, Julie Kownacki, Ernst Kraemer, Mary Kumar, Ram Kumar, Dale Labossiere, Johnathan Lahr, Dick Leban, Gregg Lebovitz, Philip Lehman, Bob Leigh, Norbert Leser, Bruce Leverett, Kristen Levy, Steve Lewontin, Jeff Liotta, Suzanne Lipsky, Dave Lounsbury, Kevin Lynch, Tom Lyons, Mike Machutt, Dick Mackey, David Magid, Dottie Mamos, Saul Marcus, Beth Martin, Liza Martin, Sandra Martin, Tony Mason, Tony Mauro, Howard Mayberry, Bob Mayes, Raymond Mazzaferro, Janet McCann, Marll McDonald, Craig McGowan, Andy McKeen, Michael McMahon, Dave Mehaffy, Howard Melman, John Milburn, Steve Miller, Pam Millett, Dah Ming Chiu, Nathaniel Mishkin, Mamata Misra, Wayne Mock, Paul Mockapetris, Keith Morgan, Kathy Moriconi, Howard Morris, Bill Mowson, Johann Mueller, Vera Mueller, Sape Mullender, Scott Nacey, Parul Nanvanti, Roger Needham, Gerhard Neumann, Wick Nichols, Laura Norton, Scott Norton, Bridget Notzon, Gary Oden, Dave Oran, Mary Orcutt, Dave Ortmeyer, Ken Ouellette, Larry Ouellette, Phil Owens, Scott Page, Maryanne Paratore, Margo Parent, Joe Pato, Marco Pauletti, Peter Pawlita, Alan Peckham, Steve Peckham, Jean Pehkonen, Ed Perkins, Jim Perry, Dennis Phillips, Shiobhan Pigott, William Pigott, Marty Port, Jeff Prem, Hal Prince, Giovanni Rabaioli, Dan Raizen, John Raleigh, Mary Beth Raven, Ron Rebeiro, Joel Richman, Susie Richter, Davie Robinson, Larry Rose, Ward Rosenberry, Danna Rother, John Rousseau, David Royal, Donna

Ruane, Gabi Rustemeyer, Vincent Ryan, Mike Saboff, Sandy Sadowski, Rusty Sandberg, Mark Sawyer, Webb Scales, Peter Schay, Brian Schimpf, Wolfgang Schmid, Heinrich Schmidt, Gary Schmitt, Tom Sgouros, Ellen Sharp, Diane Sherman, Margie Showman, Chi Shue, Bob Sidebotham, Al Simons, Hermi Singh, David Skeen, Paula Slotkin, Bill Sommerfeld, Alfred Spector, Dave Stephenson, Dawn Stokes, Ellen Stokes, Kevin Sullivan, Henk Tinkelenberg, David Tory, Dave Treff, Hubert Trieb, Shu-Tsui Tu, Walt Tuvell, Walter Ulrich, Ellen Vliet, Helmut Volpers, Jim Wade, Ed Wahl, Ken Walker, Susanna Wallace, Linda Walmer, Una Walsh, Aidan Walters, Terri Warren, Peter Weinberger, Doug Weir, Dave Weisman, Beatrice Weiss, Reinhard Weltrich, Alvin Wen, Eric Wertz, Wayne Wheeler, Anne Williams, Frank Willison, Jean Wilson, Elaine Wolfe, Angelika Wolff, Roger Woodbury, Julie Yarsa, Lisa Zahn, Ed Zayas, Rich Zeliff, Joseph Zrihen, and the author of this manual, Jennifer Steiner.

Chapter 1

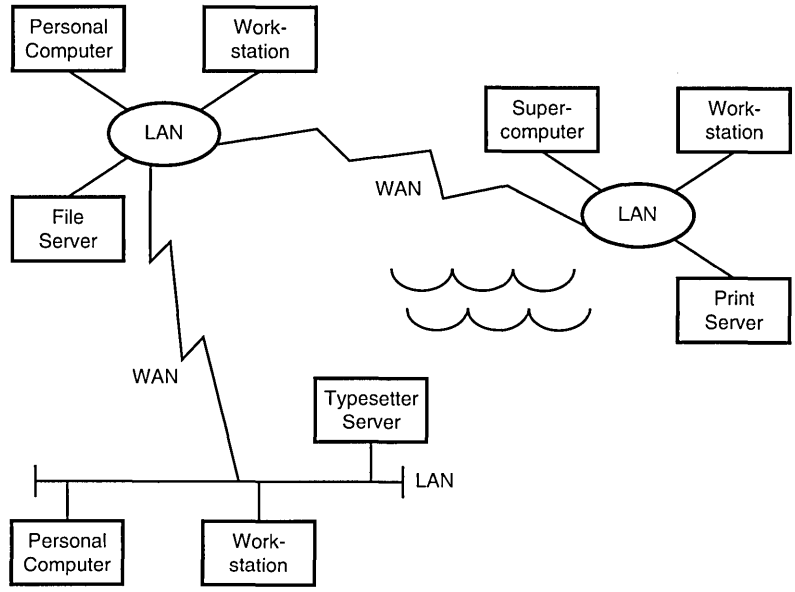
Overview of DCE

OSF's Distributed Computing Environment 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, 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.

1.1 Why Distributed Computing?

By "distributed computing" we mean computing that involves the cooperation of two or more machines communicating over a network (see Figure 1-1). 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 1–1. 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—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 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.

1.1.1 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 1-1 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:

- 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 Remote Procedure Call 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 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.)

- 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.

- DCE provides interoperability and portability across heterogeneous platforms.

  Another 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.

- 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.

- 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—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 using the same global name.

1.1.2 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:

- An office with isolated computing resources can network the computers together and use DCE for data and resource sharing.
- 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.
- 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.

- 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.
- Organizations wishing to use applications that run on DCE platforms.
- 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.
- System vendors whose customers are in any of the preceding categories.
- Organizations that would like to make a service available over the network on one system (for example, VMS), and have it accessible from other kinds of systems (for example, UNIX based workstations).

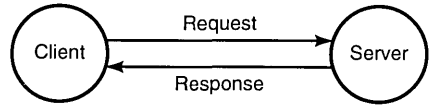
1.2 Models of Distributed Computing

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. The following subsections briefly describe each model.

1.2.1 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 1-2).

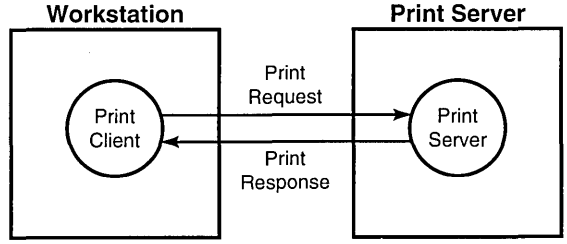
Figure 1–2. The Client/Server Model



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.

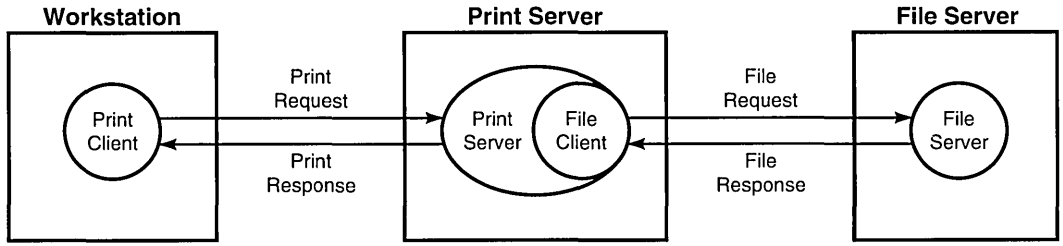
Figure 1-3 shows a workstation running the client side of a distributed print program, and a print server running the server side of the distributed program.

Figure 1–3. 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—it may ask the file server to send it a copy of the file to be printed (see Figure 1-4).

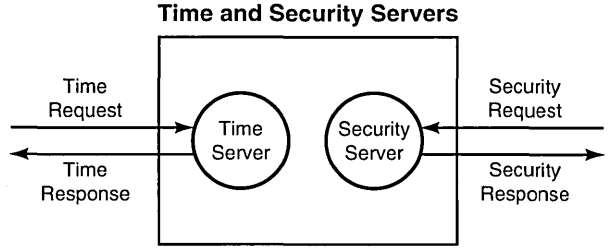
Figure 1–4. The Print Server Acting as a Client of the File Server



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 1-4, the print server, which runs on the print server machine, is acting as a client to the file server.

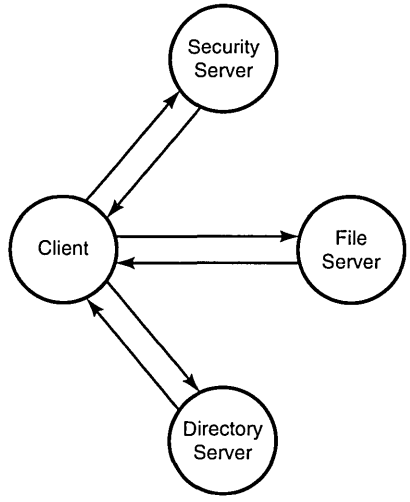
Note that it is possible for more than one server to run on a given node. For example, both a security server and a time server can run on the same machine. In this case, the given node is both the security server machine and the time server machine (see Figure 1-5).

Figure 1–5. 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 1-6.

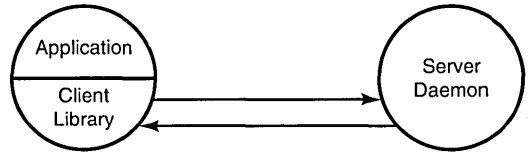
Figure 1–6. A Client Is General; Servers Are Specialized



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: 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 1-7 illustrates this distinction.

Figure 1–7. 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.

1.2.2 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 Section 3.2 of this manual for more information on remote procedure calls and DCE RPC.)

1.2.3 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—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.

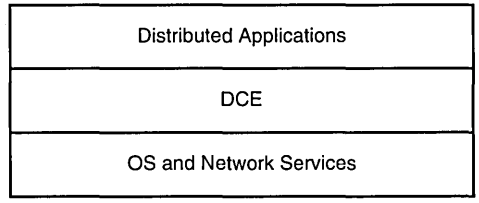
The DCE Distributed File Service is also based on the data sharing model. A DFS client maintains a cache of files that have recently been accessed by a user on the system. DFS servers distribute and revoke tokens, which represent a client's capability to perform operations on files. Through careful token 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.

1.3 Architectural Overview of DCE

OSF'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 1-8 shows DCE in relation to operating systems, network communications software, and applications software.

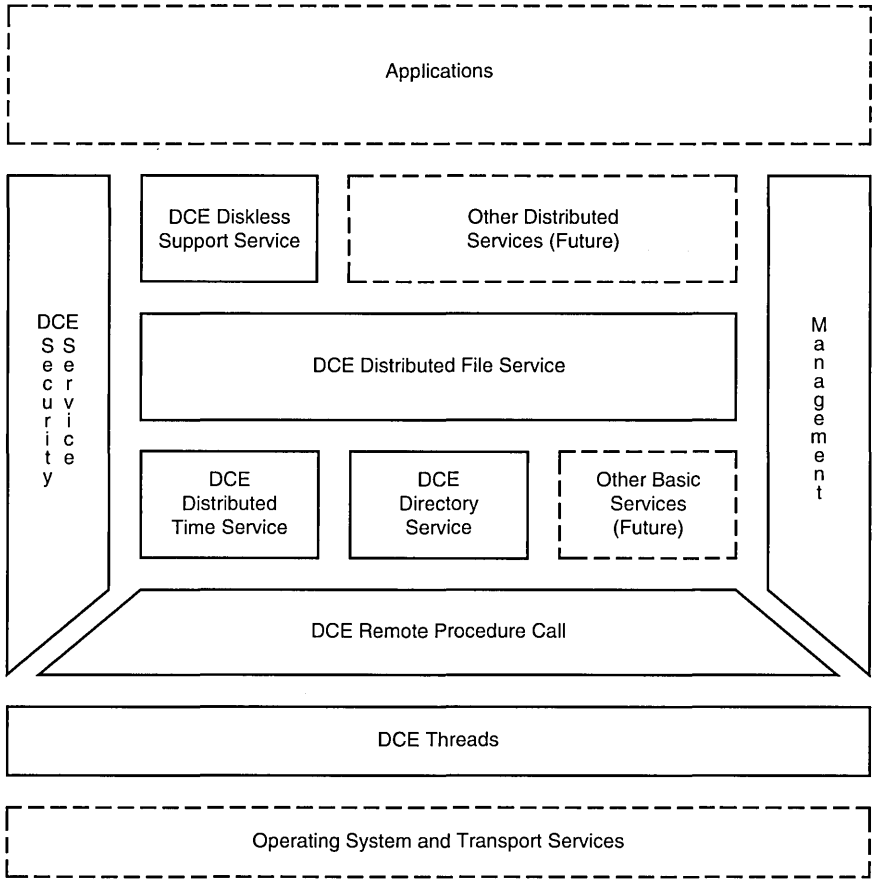
Figure 1–8. Layering of DCE and Related Software



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 1-9 shows the DCE architecture and its technology components, along with their relationship to applications, underlying system support, and placeholders for future technologies.

Figure 1–9. DCE Architecture



1.3.1 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 of this manual.

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. If the host operating system already supports threads, DCE can use that software and DCE Threads is not necessary. However, not all operating systems provide a threads facility, and DCE components require that threads be present, so this user-level threads package is included in DCE.

The **DCE Remote Procedure Call (RPC)** 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 Service (GDS), the Global Directory Agent (GDA), and a directory service programming interface. The Cell Directory Service 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 Global Directory Service implements an international standard directory service, and provides a global namespace that connects the local DCE cells into one worldwide hierarchy. The Global Directory Agent (GDA) acts as a go-between for cell and global directory services. Both CDS and GDS are accessed using a single directory service application programming interface, the X/Open Directory Service (XDS) 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 three aspects to DCE security: authentication, secure communications, and authorization. These aspects are implemented by several services and facilities that together comprise the DCE Security Service, including the Registry Service, the Authentication Service, the Privilege Service, the Access Control List (ACL) Facility, and the Login Facility.

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—communication over the network can be checked for tampering or encrypted for privacy. Finally, 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 Access Control List. 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.

The **DCE Distributed File Service (DFS)** 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 using the same name. The Distributed File Service 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.

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 also offers **Diskless Support Service,** which provides the tools that allow a diskless node to acquire an operating system over the network, obtain configuration information, connect to DFS to obtain the diskless node's root file system, and perform remote swapping. When these tools are incorporated into the client's operating system and hardware, the diskless node can operate in a DCE environment.

The **Management** block shown in Figure 1-9 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.

1.3.2 Organization of a Distributed Computing Environment

This section introduces the concept of a DCE cell, and gives a brief summary of how different machines participating in a Distributed Computing Environment are organized.

A group of DCE machines that work together and are administered as a unit is called a **cell**. For example, imagine an organization comprised 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 "Distributed Computing Environment" (or "DCE environment") is a group of one or more DCE cells that can communicate with each other. A cell becomes a part of a DCE environment when it obtains access to one or more global directory services in which the other cells in the environment are registered.

Going back to the example, if the different departments' cells are a part of a DCE environment, 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.

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 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:

- Cell Directory Server
- Security Server
- Distributed Time Server

Other DCE servers may be present in a given DCE cell to provide additional functionality—a Global Directory Agent 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, the special functions of the Local File System, and possibly Diskless Support Service. (See Chapter 2 of this manual for more detailed information on DCE cell configuration.)

1.3.3 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. The Distributed File Service uses Threads, RPC, Directory Service, Distributed Time Service, and Security Service in providing its file service.

In general, the DCE components shown higher in the DCE Architecture (see Figure 1-9) make use of the components shown lower in the architecture. For example, DCE Threads is used by most other DCE components, but 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.

1.3.4 Relationship of DCE to Network and System Services

As shown in Figure 1-8, 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.

1.3.4.1 Network Services

In general, DCE is layered over a transport level service, such as UDP (User Datagram Protocol), TCP (Transmission Control Protocol), or ISO TP0-TP4 transport protocols, which is accessed through a transport interface, such as sockets or XTI (X/Open Transport Interface). 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 OSF DCE 1.0 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).

1.3.4.2 Operating System Services

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

- Multitasking
- Timers
- Local interprocess communications
- Basic file system operations (VFS layer)
- Memory management
- Local security mechanisms (if appropriate)
- Threads (or the ability to use DCE Threads)
- General system utility functions

1.3.4.3 DCE Reference Implementation Dependencies

The previous two subsections listed assumptions made by the DCE architecture. The OSF DCE 1.0 reference implementation contains additional dependencies on the operating system and network, which are specific to the implementation. These include the use of Internet Protocol and socket networking services, and UNIX operating system facilities.

Chapter 2

DCE Configuration

Chapter 1 gave some examples of organizations that could benefit from a Distributed Computing Environment. The examples showed that 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.

2.1 Introduction to DCE Configuration

A Distributed Computing Environment (or DCE environment) consists of machines that communicate over a network and run DCE software. The machines serve different functions and therefore run different configurations of DCE software. There are three basic types of machines in a DCE environment:

- DCE User Machine

  A 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.

- DCE Administrator Machine

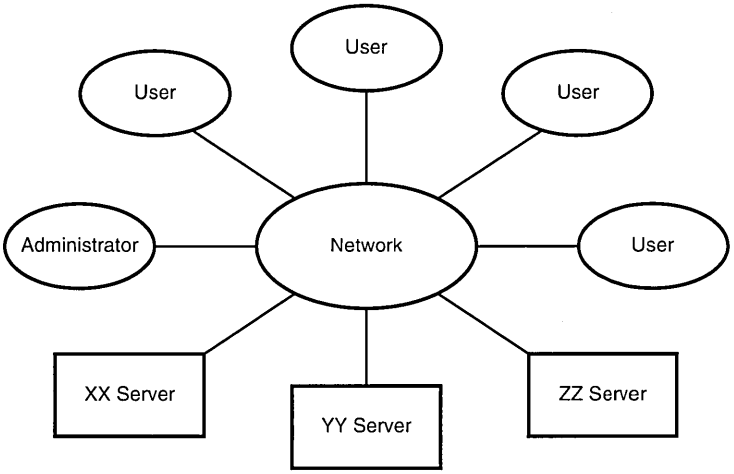
  A DCE Administrator Machine contains DCE software that enables an administrator to control servers running in the environment. A typical example is the DCE system administrator's workstation.

- DCE Server Machine

  A 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.

Figure 2-1 shows an example of a DCE environment containing the three different kinds of DCE machines.

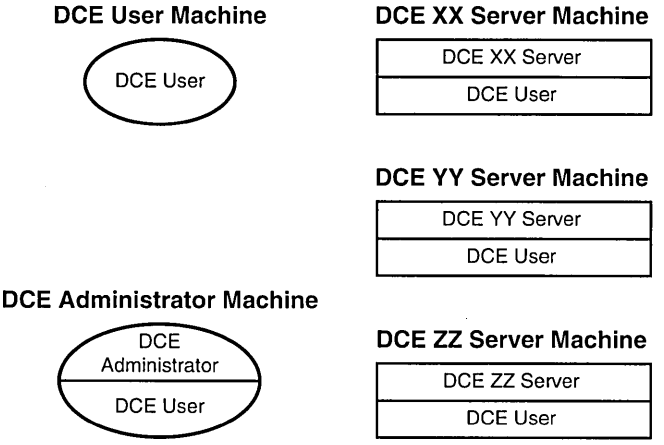
Figure 2–1. Types of DCE Machines



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. The DCE User runs on all three types of DCE machines. The software necessary for an administrator to control DCE servers remotely is the ''DCE Administrator'' software. The DCE Administrator runs on DCE Administration Machines, along with DCE User software.

Finally, 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 2-2 summarizes the DCE software that runs on different kinds of DCE machines.

Figure 2–2. DCE Machines and Their Software



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

2.2 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.

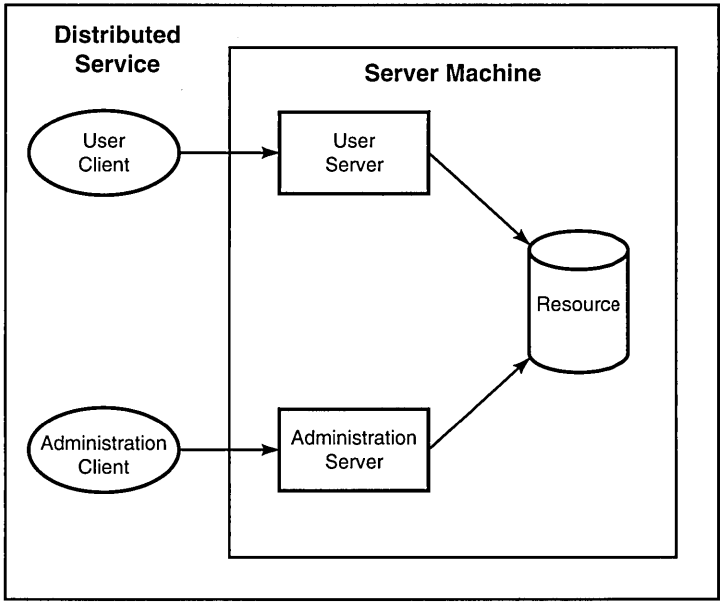
First, each DCE service can be divided into two general categories of functionality, user and administration. The *user* functionality is the service provided to its users; for example, reading a file or searching a database. The *administration* functionality allows administrators to manage the server;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

- User Client
- User Server
- Administration Client
- Administration Server

As shown in Figure 2-3, the User Client communicates over the network with the User Server, and the Administration Client communicates over the network with the Administration Server.

Figure 2–3. 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 **rgy_edit** program, which administrators use to modify data in the security database. It 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, the Distributed Time Service, the Security Service, the Distributed File Service, and the Diskless Support Service, can all be divided roughly into these four configuration components.

DCE Threads and DCE RPC are separate configuration components. They help to implement the communications between machines, so they must be present on every DCE machine, whether the machine acts as a client or a server.

Section 2.3 describes how machines participating in a DCE environment are configured, using various combinations of configuration components. Section 2.4 describes how DCE cells are configured, using various combinations of DCE machines.

2.3 DCE Machine Configuration Examples

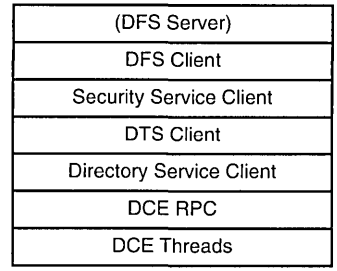
DCE machine configurations fall into three general categories: client machines, administrator machines, and server machines.

2.3.1 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 2-4). Part of this software may be present in the form of libraries linked with DCE application software.

Figure 2–4. 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.

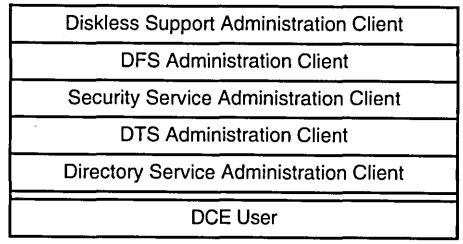
We call the software configuration of a typical DCE User Machine the ''DCE User'' software. In summary, the DCE User contains

- DCE Threads and DCE RPC
- User Client configuration components of each DCE service
- DFS Server software (optional)

2.3.2 DCE Administrator Machine Configuration

A DCE administrator's workstation is configured with the client sides of DCE administration programs, to enable the administrator to control servers remotely. This configuration contains the Administration Client software for each of the DCE services. It also contains the DCE User software, since the Administrator Machines act as User Clients as well as Administration Clients (see Figure 2-5).

Figure 2–5. DCE Administrator Machine Configuration

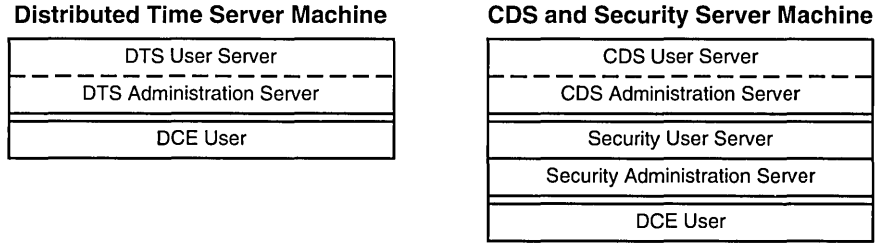


2.3.3 DCE Server Machine Configuration

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 node; two or more types of servers can run on a single machine. Figure 2-6 shows the configuration of a Distributed Time Server machine and the configuration of a second machine acting as both a CDS Server and a Security Server.

Figure 2–6. 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.

2.4 DCE Cell Configuration Examples

DCE cells consist 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, Security, and Distributed Time Servers in every DCE cell. In addition, a DCE cell can contain any combination of the remaining DCE servers—GDS, DFS, and Diskless Support Service—depending on the needs of the DCE users.

The following subsections describe these typical DCE cell configurations:

- Simple DCE Cell
- DCE Cell with DFS File Server Machine
- Connected DCE Cell
- Multicell Configurations

2.4.1 Simple DCE Cell

Figure 2-7 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; it runs the DCE Administrator software in addition to the DCE User software. 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 2–7. Simple DCE Cell Configuration

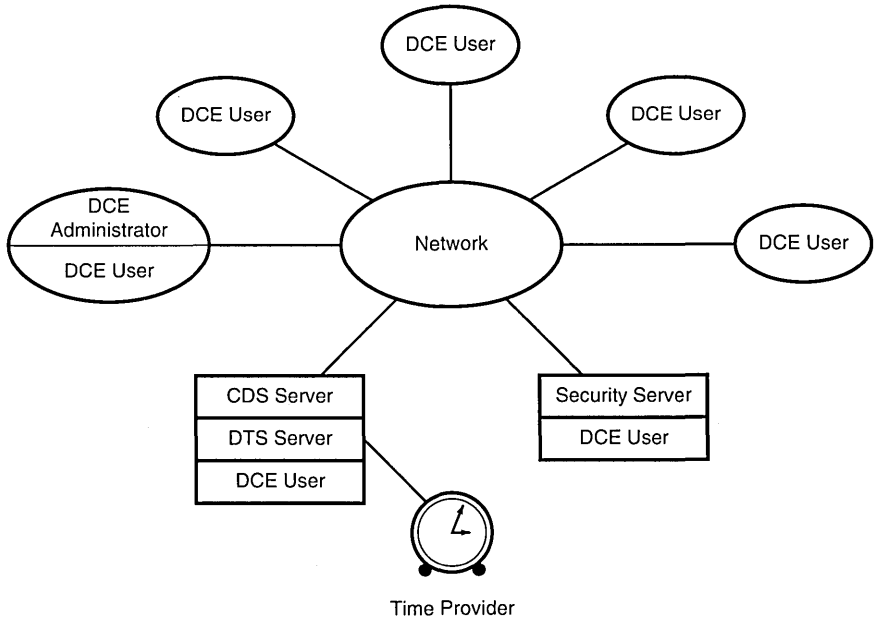
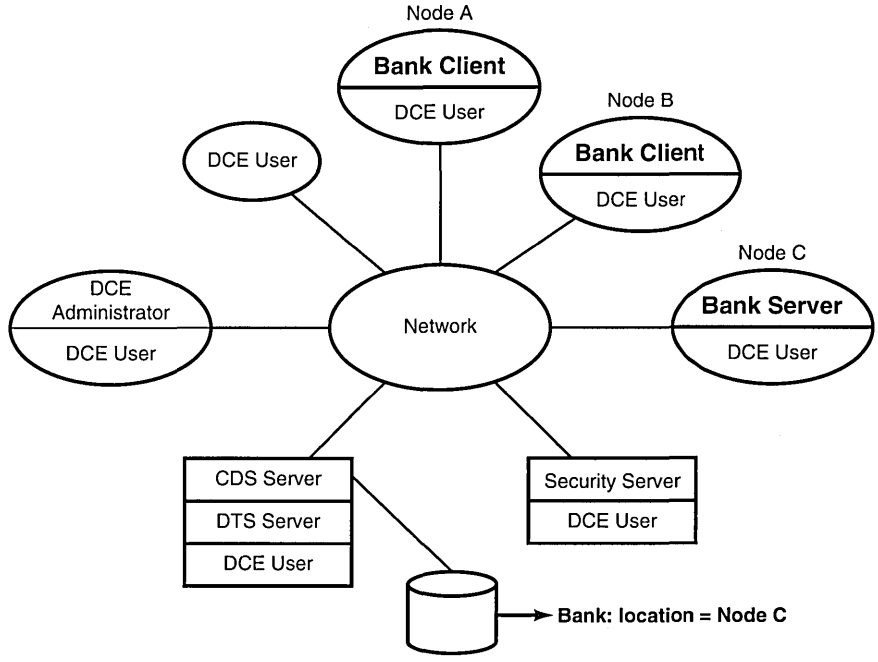


Figure 2-8 shows the same simple DCE cell, this time with a DCE application running in it. Node C is offering the Bank Service, and Nodes A and B have the client code for accessing the Bank Service. The Bank Server has registered itself in the Cell Directory Service so the Bank Clients are able to locate it.

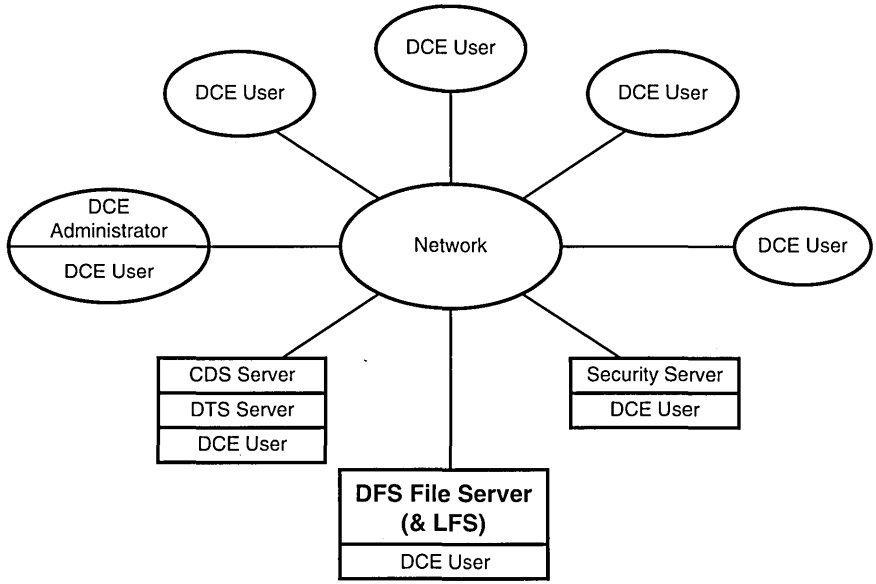
Figure 2–8. DCE Application in Simple Cell



2.4.2 DCE Cell with DFS

In order to have full Distributed File Service support, including DCE's Local File System, a DCE cell can contain one or more DFS File Server machines (see Figure 2-9). As mentioned in Section 2.3.1, the DCE User is equipped to act as a DFS client, and may 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.

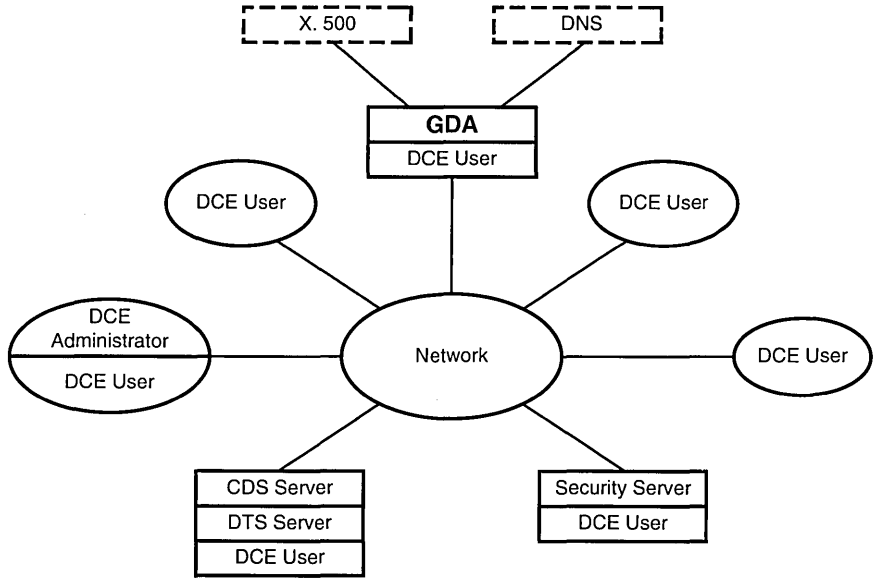
Figure 2–9. Simple Cell Plus Distributed File Server



2.4.3 Connected DCE Cell

An organization may wish to communicate with other DCE cells, or with systems outside of DCE. One way this can be accomplished is through the DCE Global Directory Service, an implementation of the X.500 directory service standard. DCE also supports the use of the Domain Name Service (DNS) as a global directory service. The cell's CDS communicates with CDS servers in foreign cells with the help of an intermediary, the Global Directory Agent. When a Global Directory Agent machine is added to a DCE cell, the nodes in the cell will be able to contact other systems using X.500 or DNS. Figure 2-10 shows the simple DCE cell with a Global Directory Agent added to it.

Figure 2–10. Cell Connected via Global Directory Agent



Finally, if a cell contains a Global Directory Server, it can not only access the X.500 namespace through the GDA, but it can also own and administer a portion of that namespace in the GDS. For more information on the Global Directory Service and Cell Directory Service, see Section 3.3 of this manual.

2.4.4 Multicell Configurations

An organization may decide to have a DCE configuration that consists of more than one cell. For example, the organization might consist of several departments, each wanting to have administrative control of its resources. In this case, each department in the organization could administer its own cell. This results in slightly more total administrative overhead than a single, large cell, but the local administrative control of each cell may be worth the tradeoff. If the cells were connected through a global directory service, as described in the previous section, then the users of one cell will be able to access the resources in another cell, if they are authorized.

Chapter 3

DCE Technology Components

The OSF Distributed Computing Environment comprises several technology components:

- DCE Threads
- DCE Remote Procedure Call
- DCE Directory Service
- DCE Distributed Time Service
- DCE Security Service
- DCE Distributed File Service
- DCE Diskless Support Service

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 Application Programming Interfaces (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.

This chapter contains one section devoted to each of the technology components (Sections 3.1 through 3.7). Each of these sections starts with an overview of its technology, along with a description of the pieces that comprise the technology. The sections then describe their technologies from the perspective of different types of users: 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 describe important benefits or features of the particular technology.

The last section of this chapter, Section 3.8, gives an example of a very simple distributed application, describing the process for developing, installing, and running it.

3.1 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 pre-existing 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.

3.1.1 What is DCE Threads?

DCE Threads is a user-level (nonkernel) threads library based on the pthreads interface specified by POSIX in their 1003.4a standard (Draft 4). It consists of an API that gives programmers the ability to create and manipulate threads, as described in Section 3.1.3. The other technology components of OSF's Distributed Computing Environment 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.

DCE Threads is designed for compatibility with existing operating systems that deal with processes rather than threads, and libraries that are not reentrant (that is, not written to handle multiple threads executing within them at the same time). This compatibility is provided through the use of "jacket" routines, which are used in conjunction with existing libraries, and modified operating system calls. Since messages from the outside world (such as interrupts and signals) have traditionally been addressed to a process, rather than a specific thread in a process, this interaction must be modified as well. For further information on the way DCE Threads interacts with other software, see the chapters on threads in the *OSF DCE Application Development Guide.*

3.1.2 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.

3.1.3 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.

3.1.3.1 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.

3.1.3.2 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:

- Scheduling priorities
- 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 they are running under.

DCE Threads offers three scheduling policies:

- First-In, First-Out (FIFO)

  In FIFO scheduling, 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.

- Round-Robin (RR)

  With round-robin scheduling, 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 round-robin until they are all finished or blocked, and so on.

- Default

  In the default scheduling, 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 round-robin scheduling, in which it is possible for higher priority threads to prevent lower priority threads from running at all.

3.1.3.3 Thread Communication and Synchronization

Threads communicate through shared variables—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:

- Mutual exclusion objects (mutexes)
- Condition variables
- 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 ''mutual exclusion'' or ''mutex'' name.

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.

3.1.3.4 DCE Threads Exceptions

DCE Threads provides two ways to obtain information about the results of a threads call. One way is specified by the POSIX P1003.4a (pthreads) draft standard—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.

3.1.4 DCE Threads Administration

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

3.1.5 Additional Information on DCE Threads

For additional information on DCE Threads, see the following:

- The DCE Threads chapters of the *OSF DCE Application Development Guide*
- The **(3thr)** reference pages of the *OSF DCE Application Development Reference*
- The POSIX P1003.4a/Draft 4 *Threads Extension for Portable Operation Systems* Specification
- The Implementation-Specific Addendum to the POSIX P1003.4a/Draft 4 Specification

3.2 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 Remote Procedure Call (RPC).

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:

- Fragmentation and reassembly of messages
- Handling different data formats (such as byte ordering) between different machines
- Using a directory service to find message recipients
- 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 Remote Procedure Call technology component. They describe the components that comprise the technology, and how DCE RPC is seen from the end user's, programmer's, and administrator's perspective, 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.

3.2.1 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 comprise the DCE RPC are as follows:

- The Interface Definition Language (IDL) and its Compiler

  An RPC interface is described in DCE IDL. The IDL file is compiled into object code using 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.

- 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.

- 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 Section 3.5.4 for more information on Authenticated RPC.)

- 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.

- The RPC Daemon

  The RPC daemon (**rpcd**) is a program that runs on every DCE machine. It is an RPC-specific name server, which manages a database that maps RPC servers to the transport endpoints (in IP, the ports) that the server is listening for requests on.

- The RPC Control Program

  The RPC control program (**rpccp**) is a tool for administering **rpcd**. It also allows an administrator to access RPC data in CDS.

- UUID Facilities

  These are ancillary commands and routines for generating Universal Unique Identifiers (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.

3.2.2 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

- The availability of distributed applications built using RPC
- The ability to use remote resources accessed via RPC

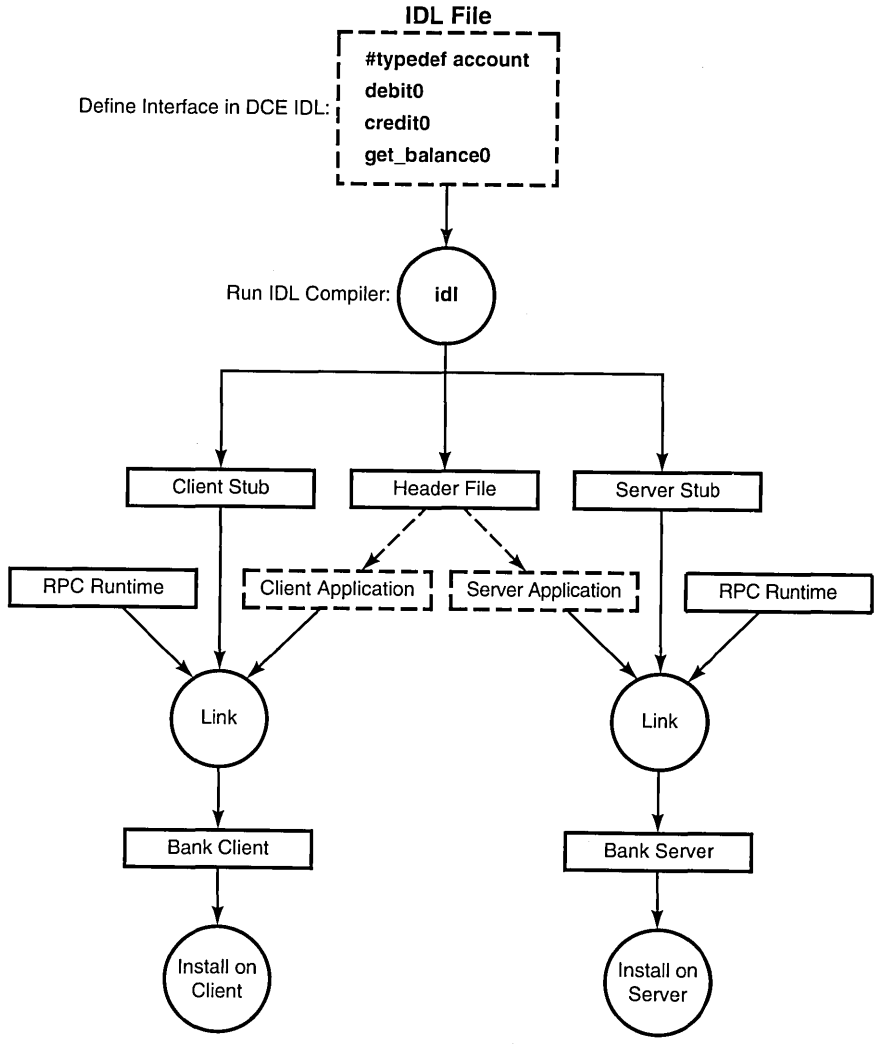
An end user who is browsing through the namespace may also notice the names of RPC-based servers, since these servers advertise themselves to their clients through the DCE Directory Service.

3.2.3 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 Section 3.8 of this manual. For a more detailed description of the DCE RPC programming process, see the introductory chapters and the RPC chapters of the *OSF DCE Application Development Guide*.

Figure 3-1 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 3–1. DCE RPC Programming Process



3.2.3.1 The IDL File

First, the application programmer defines the RPC interface, and associated data types, using the DCE Interface Definition Language (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—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 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 using the IDL compiler. The compiler produces output either in a conventional programming language, which is the C language in the DCE offering, or in 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.

3.2.3.2 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.

3.2.3.3 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.

3.2.3.4 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.

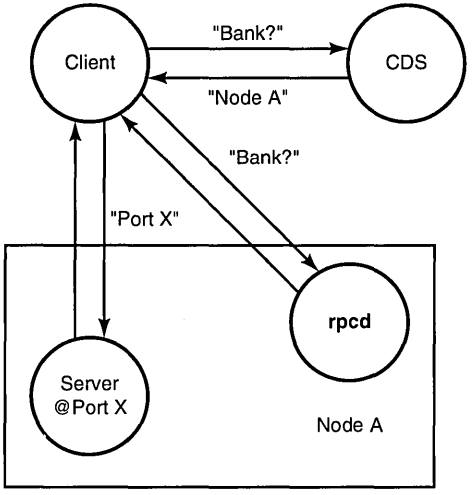
The DCE programmer does not make calls directly to CDS; binding is supported by the Name Service Independent (NSI) API, an RPC-specific name service layer. Calls to this library are made by the client side of an application in order to look up binding information for a server based on various criteria,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

3.2.3.5 The RPC Daemon

There are two parts to a server's location: the address of the machine it resides on, and the 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 the Cell Directory Service. The network endpoint, however, can change each time the server process is started up. Instead of making frequent changes to CDS to update a server's endpoint address, DCE RPC uses a specialized type of directory service, the RPC daemon, or **rpcd**. When a server starts up, it registers its process address with **rpcd**.

Every machine that runs an RPC server also runs an **rpcd**. The **rpcd** 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 **rpcd** process running on that same machine. It can then ask the **rpcd** process for the network endpoint of the server process. This process is shown in Figure 3-2 (read the messages, show in quotes, in clockwise order).

Figure 3–2. Client Finds Server Using CDS and RPC Daemon



3.2.4 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 up. As described in the previous section, the server registers its (dynamically assigned) listening endpoint with **rpcd**. 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 an RPC 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.

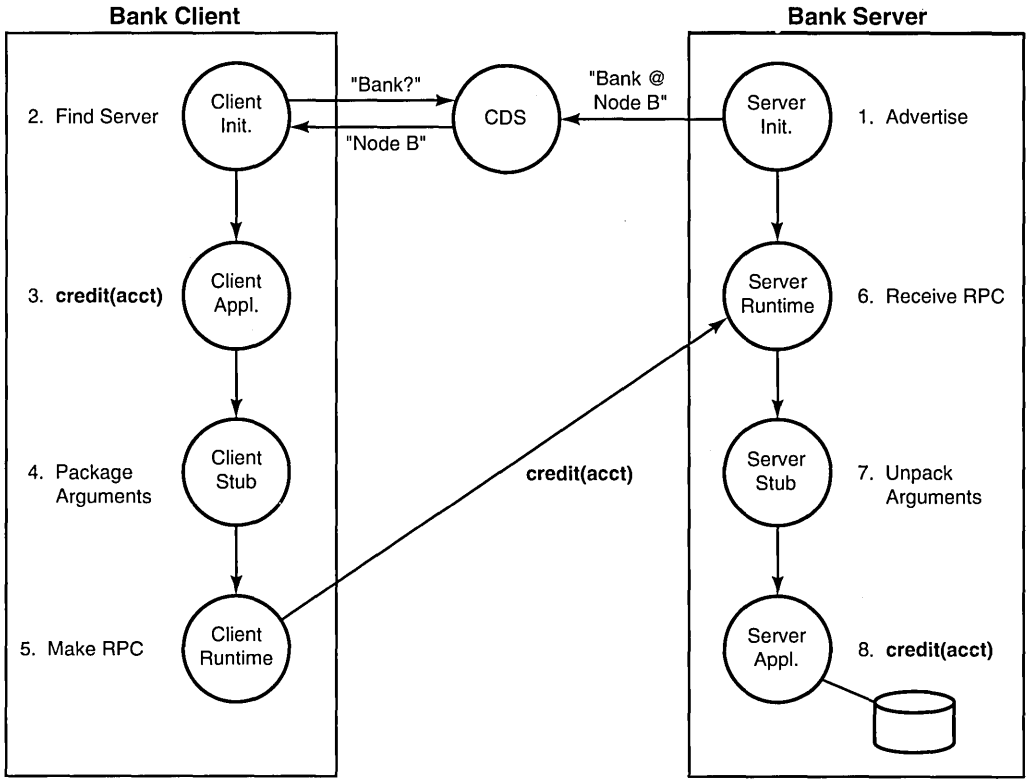
A management program, **rpccp,** allows an administrator to control the **rpcd** and administer RPC information in the namespace.

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

3.2.5 How It 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 3-3. (This description is somewhat simplified. The use of **rpcd** is not shown.)

Figure 3–3. 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 the Cell Directory Service (see Point 1 in Figure 3-3). 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—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.

3.2.6 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.

- 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.

- 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.

- 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.

- 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.

- 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.

3.2.7 Additional Information on DCE RPC

For additional information on DCE RPC, see the following:

- The RPC chapters of the *OSF DCE Application Development Guide* and the *OSF DCE Administration Guide*
- The **(1rpc)** and **(3rpc)** reference pages of the *OSF DCE Application Development Reference*
- The **(5rpc)** and **(8rpc)** reference pages of the *OSF DCE Administration Reference*

3.3 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. Sections 3.3.2 through 3.3.4 describe each of the DCE Directory Service components—the Cell Directory Service, the Global Directory Service, the Global Directory Agent. Section 3.3.5 describes the Directory Service application programming interface.

3.3.1 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—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.

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.

3.3.1.1 Overview of Directory Service Components

There are three components that together comprise the DCE Directory Service:

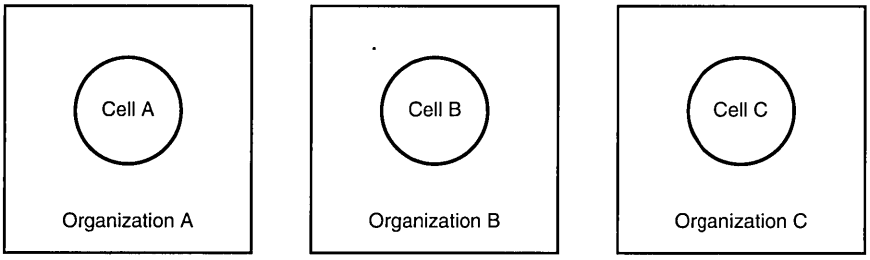
- The DCE Cell Directory Service (CDS)
- The DCE Global Directory Service (GDS)
- The DCE Global Directory Agent (GDA)

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. The 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. 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 Cell Directory Server in each DCE cell.

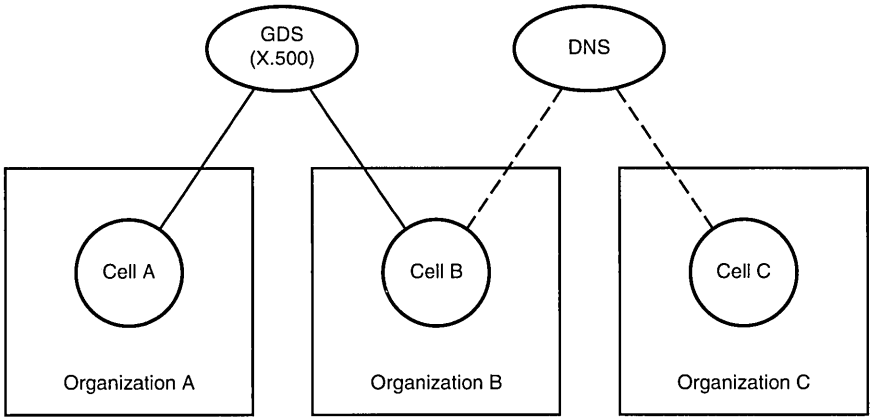
DCE Global Directory Service. The Global Directory Service is a distributed, replicated directory service based on the CCITT X.500/ISO 9594 international standard. It is used when looking up a name outside of the local DCE cell. In particular, it acts as the high-level connector that allows independent cells to find out about and interact with one another. GDS interworks with other X.500 implementations, and can therefore participate in the worldwide X.500 directory service. Figure 3-4 shows three organizations, each with its own DCE cell.

Figure 3–4. Three One-Celled Organizations



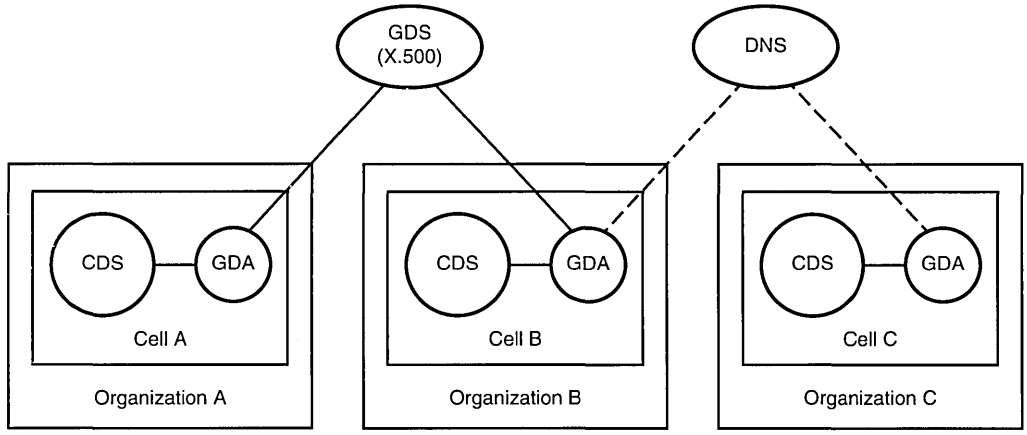
The Global Directory Service can act as a higher level directory service to connect cells, as shown in Figure 3-5. DCE supports the use of a second standard directory service, the Domain Name Service (DNS), which is widely used in the Internet community. It, too, can act as a higher level connector of DCE cells.

Figure 3–5. GDS and DNS Connect DCE Cell Namespaces



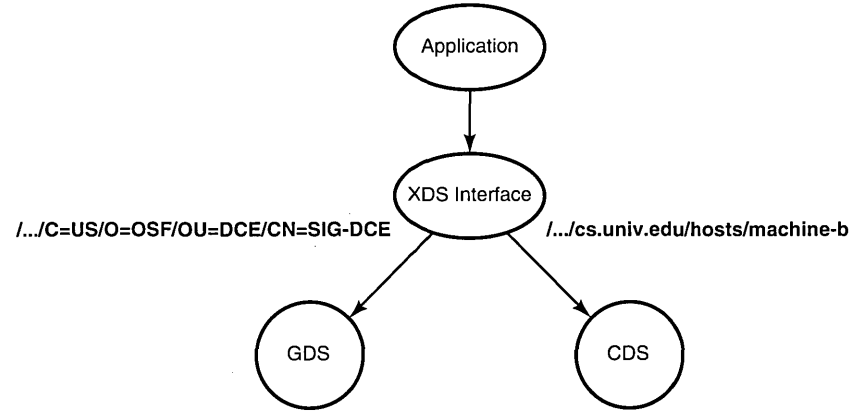
DCE Global Directory Agent. The Global Directory Agent is the intermediary between a cell's CDS and the rest of the world. It takes a name that cannot be found in the local cell and finds the foreign cell in which the name resides, using GDS or DNS, depending on where the foreign cell is registered. Figure 3-6 gives a functional picture, including the use of GDAs, of the configuration shown in Figure 3-5.

Figure 3–6. Use of Global Directory Agents



DCE Directory Service Application Programming Interface. DCE programmers use the X/Open Directory Service (XDS) application programming interface 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 the Global Directory Service or to the Cell Directory Service (see Figure 3-7). XDS uses the X/Open Object Management (XOM) application programming interface to define and manage its information.

Figure 3–7. XDS: Interface to GDS and CDS



3.3.1.2 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.

Figure 3-8 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=OSF/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 3–8. Four Cells in DCE Global Namespace

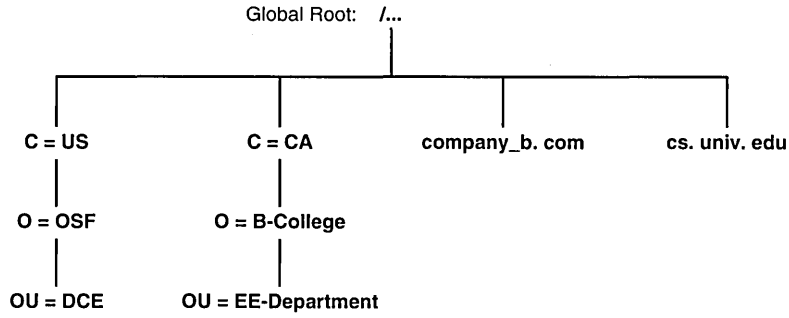
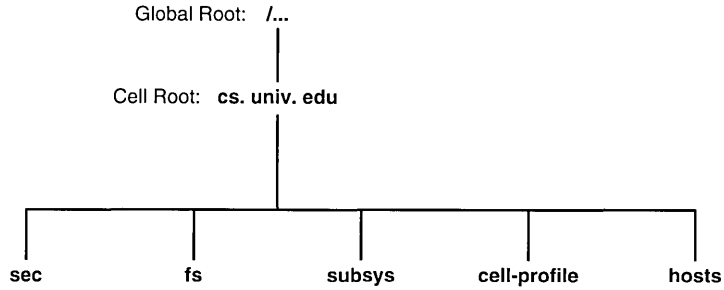


Figure 3-9 shows the top of a typical DCE cell namespace. It contains an entry for security information, an entry for the cell's DFS file system, an entry for subsystems such as DCE services, an RPC Profile entry, and an entry for hostnames. (See the first module of the *OSF DCE Administration Guide* for more information on the cell namespace.)

Figure 3–9. Top of a Typical DCE Cell Namespace



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

/.../C=US/O=OSF/OU=DCE/sec/principals/snowpaws
/.../C=US/O=OSF/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=OSF/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 above 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=OSF/OU=DCE/CN=SIG-DCE**. This name could be used, for example, for an electronic mail list.

3.3.1.3 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, the Global Directory Agent, and the GDS server, if the cell has one.

3.3.1.4 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 Universal Unique Identifiers (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 *OSF DCE 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, 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 *OSF DCE Application Development Guide*.

3.3.1.5 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:

- The Security Service Database

  The Security Service 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**.

- The Fileset Location Service in DFS

  The Fileset Location Service keeps 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**.

3.3.2 DCE Cell Directory Service

One of the two directory services underlying the XDS API is the DCE Cell Directory Service (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.

3.3.2.1 What Is CDS?

The DCE Cell Directory Service is made up of several components, including the CDS Server, CDS Clerk, and CDS administration programs.

- CDS Server

  The CDS Server runs on a node containing a database of directory information. It responds to queries from clients by accessing the database. (A CDS database is called a **clearinghouse**.)

- CDS Clerk

  The 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.

- CDS Administration Programs

  Two administrative programs are included in the CDS technology—the CDS Namespace Browser and the CDS Control Program. The CDS Namespace Browser, which is used by CDS administrators as well as end users, is a CDS client application that allows the user to inspect the cell's namespace. The CDS Control Program, **cdscp**, enables administrators to control CDS Servers.

3.3.2.2 The CDS Database

CDS information is contained in three types of data elements—directory entries, directories, and clearinghouses.

- 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.

- 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 may also have child directories.

- 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.

3.3.2.3 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:

- Master Replica
- 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:

- Immediate Propagation
- 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.

In some cases, it is not necessary for copies to be updated immediately. Sometimes it is not even possible, since a server holding a copy may be unavailable to receive updates. In these cases, the other consistency mechanism, **skulking**, can be used. 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. CDS allows the client application to bypass the Clerk's cache and go directly to the CDS Server for information, when the application wants to make sure it has the latest information.

3.3.2.4 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 Namespace Browser.

3.3.2.5 Programming with CDS

Programmers can access CDS through XDS (see Section 3.3.5). They also use CDS indirectly when they use the Name Service routines of the RPC API.

3.3.2.6 CDS Administration

CDS administrators use the CDS Control Program to administer CDS, and the CDS Namespace Browser to inspect its data. Some administrative tasks include determining the number of CDS Servers in the cell, and specifying replication and update of CDS data.

3.3.2.7 Additional Information on CDS

For additional information on CDS, see the following:

- The CDS chapters of the *OSF DCE Administration Guide*
- The **(8cds)** reference pages of the *OSF DCE Administration Reference*

3.3.3 DCE Global Directory Service

The DCE Global Directory Service (GDS) is a directory service implementation based on the international standard CCITT X.500/ISO 9594. When present in a DCE cell, the GDS can serve two basic functions. First, it can participate in a high level, possibly worldwide directory service tying together independent DCE cells. Second, it can be used as an additional directory service to CDS for storing object names and attributes in a central place.

Like the Cell Directory Service, GDS is a replicated, distributed database. The GDS database contains information that can be distributed over several GDS servers. In addition, copies of information can be stored in multiple GDS servers, and the information can also be cached. The unit of replication in GDS is the directory entry (whole subtrees can also be specified).

The GDS directory is structured differently from CDS, and names are also different in that they are typed, as described later. Programmers can access both DCE Directory Services, however, using the X/Open Directory Service API (see Section 3.3.5 for a description of XDS).

The following subsections describe the GDS components, possible GDS configurations, the GDS database and names, an overview of how GDS works, and the relationship of DCE GDS to underlying network services and international standards.

3.3.3.1 What Is GDS?

There are several components that work together to provide the DCE Global Directory Service:

- The Directory System Agent (DSA)

  This process runs on the GDS Server machine and manages the GDS database. It is the server side of GDS. In order to handle simultaneous requests from different users, the GDS Server machine can run several DSA processes.

- The Directory User Agent (DUA)

  The DUA is a library that implements the GDS client; this library is present on all GDS client machines.

- The Directory User Agent Cache

  This process keeps a cache of information obtained from DSAs. One DUA Cache runs on each client machine and is used by all the users on that machine. The DUA Cache contains copies of recently accessed object entries and information about DSAs. The programmer specifies which information should be cached, and it is possible to bypass the DUA Cache to obtain information directly from a DSA. This is desirable, for example, when the user wants to make sure the information obtained is up-to-date.

- The C-Stub and S-Stub

  The C-Stub process runs on each client machine and manages communications with DSAs. It implements the upper layers of the ISO protocol stack (see Section 3.3.3.6). Its function is similar to the RPC Runtime (GDS uses OSI protocols instead of DCE RPC). The S-Stub is similar to the C-Stub, except it runs on the server machine and manages its communications with DUAs and other DSAs.

- The Shadow Update and Cache Update Processes

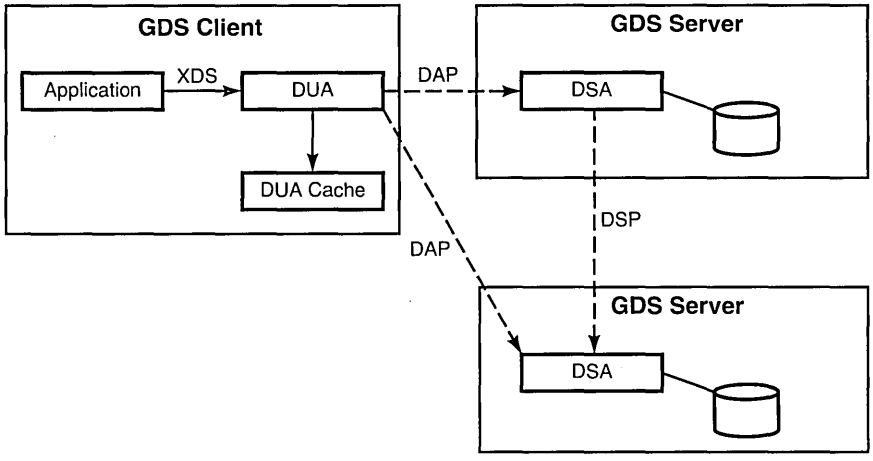
  Unlike the processes listed previously, which run continuously, the processes for updating replicas in DSAs and DUA Caches run as needed and then terminate. The shadow update process runs on the GDS Server machine; the cache update process runs on GDS client machines.

- The GDS Administration Programs

  DCE GDS provides programs for administering its service. One, **gdssysadm**, supports administration of the local GDS installation, such as configuration, server activation, and backup. The **gdsditadm** program supports remote administration of the contents of a GDS database. Finally, the **gdscacheadm** program supports the administration of the contents of the local DUA cache.

Figure 3-10 shows the interaction between the Directory Service application, the XDS interface, and the GDS client and server. The GDS client and server use the Directory Access Protocol (DAP) to communicate. The GDS servers use the Directory System Protocol (DSP) to communicate with one another. DAP and DSP perform functions similar to the function that the DCE RPC protocols perform in other DCE services. The DAP and DSP protocols are defined in the X.500 standard, enabling worldwide interoperability among directory services.

Figure 3–10. GDS Components



3.3.3.2 GDS Configurations

A GDS machine can be configured in two ways:

- Client Only

  A node can contain only the client side of GDS. This node can access remote DSAs and cache some information in the DUA cache.

- Client/Server

  A machine can be configured with both the GDS client and server. This is the typical configuration for a machine acting as a GDS server. This configuration can be useful even if a node acts mainly as a client because the DSA can be used as a larger, more permanent cache of information contained in remote DSAs.

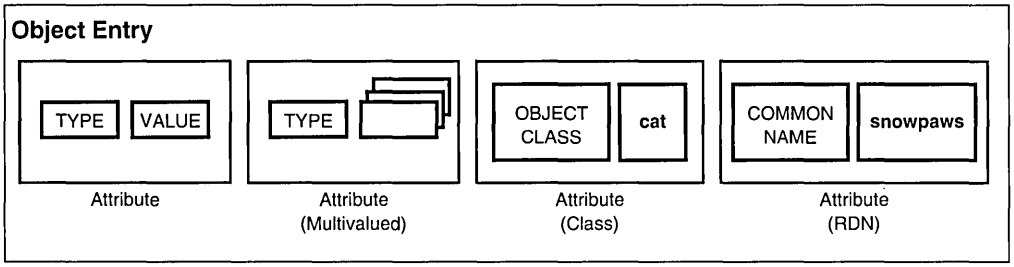
3.3.3.3 The GDS Database

The GDS database is a hierarchical, object-oriented database. The DCE Release 1.0 GDS implementation uses the C-ISAM database software. The information comprising the Global Directory Service takes the following forms:

- Object Entry

  An **object entry** is an entry in the database that names and describes an object, such as a person, machine, or server. It consists of one or more attributes, and each of the attributes has a type and a value. For example, an attribute type might be COMMON NAME (or CN) and the value might be snowpaws; another attribute might be type MACHINE ADDRESS and the value might be 100.100.1.177. Some attributes may have more than one value. Each object entry has an attribute of type OBJECT CLASS, and its value specifies what the object's class is, which determines what other attributes the object entry has. The name of an entry consists of one or more of its attributes (see Figure 3-11).

Figure 3–11. GDS Object Entry



- Relative Distinguished Name (RDN)

  The name attribute of an object contains the object's Relative Distinguished Name. An RDN contains both the type and value of the naming attribute; for example, "CN = snowpaws" or "MACHINE NAME = MachineA." (In DCE Directory Service notation, the type and value of an attribute are separated by an equal sign.)

- Distinguished Name (DN)

  The Distinguished Name is the concatenation of the object's RDN and the RDNs of all its ancestors in the GDS naming hierarchy, like a full pathname for a file in a UNIX file system. An example of a DN might be **/.../C=US/O=OSF/OU=DCE/CN=snowpaws**. (In DCE Directory Service notation, the RDNs are separated by slashes.)

- Directory Information Base (DIB)

  The Directory Information Base consists of all the object entries in all the Directory Service Agents in GDS.

- Directory Information Tree (DIT)

  The Directory Information Tree is the structure of the GDS namespace; it determines the hierarchy of GDS names. For example, the DIT might specify that the only entries that can come directly under the DIT root are entries describing countries, such as **/.../C=US** or **/.../C=JP**.

- Directory Schema

  The Directory Schema contains structuring rules for the GDS information. This includes object classes, their attributes, and their syntax.

- GDS Access Control Lists

  Security in GDS is not integrated with the DCE Security Service. It is based on Access Control Lists, but GDS ACLs are different from other DCE ACLs. Each object entry has an ACL associated with it. It specifies permission to access the object's attributes. The attributes can be classified as PUBLIC, STANDARD, or SENSITIVE. The object's ACL grants a user or group of users five different types of permission: modify PUBLIC attributes, read or modify STANDARD attributes, read or modify SENSITIVE attributes. When a new object entry is created in the GDS directory, it inherits the security characteristics of its parent entry by default. An object entry's ACLs are attributes of the object entry.

3.3.3.4 How GDS Works

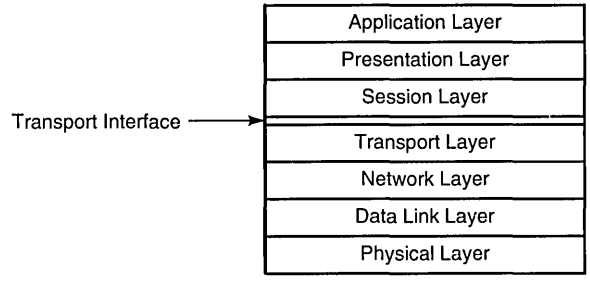
When an application program makes a Global Directory Service call using the XDS API, the call is handed to the DUA library. The DUA first looks in the DUA Cache (if specified) to see if the requested information is already available on the local node. If it is not, the DUA queries a Directory Service Agent. The DSA may have the requested information, and if it does it returns the results to the DUA. If it does not, the query can proceed in one of two ways. Either the DSA can query a different DSA on behalf of the DUA, or the DSA can return information such that the DUA can query a second DSA itself. The first method is called **chaining** and the second method is called **referral**. In either case, different DSAs are queried until the information is found. It is cached (if specified) in the DUA Cache and the results are returned to the GDS application program.

3.3.3.5 GDS and Network Services

The X.500 Directory Service standard was written to run on top of the Open Systems Interconnection (OSI) communications protocols. The OSI protocols are divided into seven layers: the Physical, Data Link, Network, Transport, Session, Presentation, and Application Layers (see Figure 3-12). The upper three layers are usually implemented as libraries that are linked together with the application process. The lower layers are part of the

operating system and their services are made available to the upper layers through a transport interface. The transport interface is the double line in Figure 3-12.

Figure 3–12. The OSI Protocol Layers



The Directory Service is an Application Layer protocol. Its specification requires the use of two other application layer service elements: ACSE (Association Control Service Element) and ROSE (Remote Operation Service Element), and of the underlying layers. ROSE and ACSE of Layer 7, and the Presentation Service of Layer 6, are implemented in GDS by the Remote Operation Service (ROS) library. The OSI Session Service (Layer 5) is implemented in GDS by the OSI Session Service (OSS) library. These layers are equivalent to the communications support supplied by the DCE RPC Runtime system, which also fills in the gap between an application and the underlying transport communications. Although GDS supplies support for these upper OSI layers, they are used only for the Directory Service and are not made available for application programmers.

DCE assumes that the system it runs on provides support for transport layer communications (either OSI transport or IP transport). The OSI protocols running above the transport layer were originally designed to run over OSI transport protocols. Many DCE systems run TCP/IP, so GDS provides the capability for running over the TCP/IP transport protocol as specified in RFC 1006.

The GDS software includes a compiler and a runtime library called MAVROS. The compiler takes specifications written in the Abstract Syntax Notation (ASN.1) and compiles them into C language code for header files and encoding/decoding routines, much as the RPC IDL compiler takes an IDL specification and compiles it into a header file and client and server stubs. MAVROS is used to encode/decode the DAP and DSP protocols and their data values.

3.3.3.6 GDS Relation to Standards

The OSI software provided in DCE is based on the following ISO standards:

- X.500/ISO 9594

  The CCITT 1988 version (Blue Book), which shares the same text as ISO Directory Standard 9594 (v1) published in 1990.

- ROSE/ACSE/Presentation/Session

  ISO 9072 (v1:1989), 8650 (v1:1988), 8649, 8823 (v1:1988), and 8327 (v2:1988) Protocol International Standards. The implementation follows EWOS agreements.

- ASN.1/BER

  The ASN.1 compiler accepts ASN.1 syntax conformant to ISO 8824 and generates routines to encode/decode data conformant to ISO 8825 Basic Encoding Rules.

The GDS software provides functional extensions to the standards in the following areas:

- Replication
- Knowledge Information Modelling and Administration
- Schema Modelling and Administration
- Subtree Administration
- Caching
- Remote Administration
- Security (Access Control)

3.3.3.7 Additional Information on GDS

Additional information on the DCE Global Directory Service and related standards can be found in the following:

- The chapters on GDS in the *OSF DCE Administration Guide*
- The **(8gds)** reference pages of the *OSF DCE Administration Reference*
- See also the standards documents listed in the previous section

3.3.4 DCE Global Directory Agent

The DCE Global Directory Agent (GDA) is the third major component of the DCE Directory Service. It 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.

3.3.4.1 What is GDA?

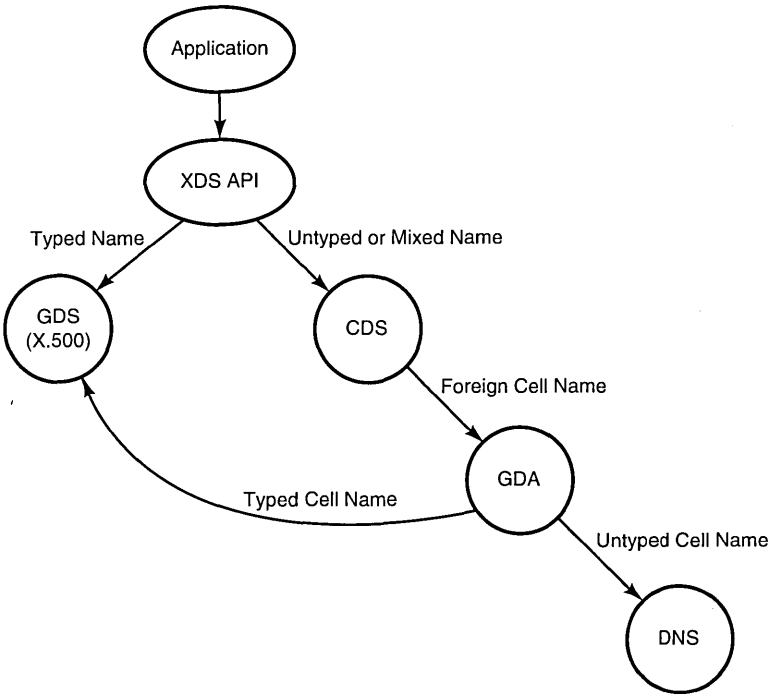
The DCE Global Directory Agent is a process that interfaces between CDS and GDS or the Domain Name Service. The GDA is not visible to the end user. Programmers access the GDA indirectly through the XDS API. GDA administration consists simply of starting and stopping the GDA process.

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.

3.3.4.2 GDA and Other Directory Service Components

Figure 3-13 shows how the GDA relates to other Directory Service components.

Figure 3–13. GDA and Other Directory Service Components



The application uses XDS to make a Directory Service call. If the name to be accessed is a typed name, such as **"/.../C=US/O=OSF/OU=DCE/CN=SIG-DCE"** , then the query is passed to the Global Directory Service. 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=OSF/OU=DCE/fs/usr/snowpaws**, then the query is passed to the Cell Directory Service. 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.

3.3.4.3 Additional Information on DCE GDA

For additional information on DCE GDA, see the GDA sections of the *OSF DCE Administration Guide*.

3.3.5 The Directory Service Interfaces

The X/Open Directory Service (XDS) and X/Open Object Management (XOM) application programming interfaces 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.

3.3.5.1 The XOM Application Programming Interface

XOM is an interface for creating, deleting, and accessing objects containing information. It is an object-oriented architecture—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.

3.3.5.2 The XDS Interface

The X/Open Directory Service (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 comprise 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.

3.3.5.3 Additional Information on XDS and XOM

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

- The XDS and XOM chapters of the *OSF DCE Application Development Guide*
- The **(3xds)**, **(4xds)**, **(3xom)**, and **(4xom)** reference pages of the *OSF DCE Application Development Reference*
- X/Open CAE Draft 1 (May 1991) Specification, API to OSI Object Management (XOM)
- X/Open CAE Draft 1 (May 1991) Specification, API to Directory Services (XDS)

3.4 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.

The DCE Distributed Time Service (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 **Coordinated Universal Time** (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—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—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.

3.4.1 What Is DTS?

There are several different components that comprise the DCE Distributed Time Service:

- Time Clerk
- Time Servers
  - — Local Time Server
  - — Global Time Server
  - — Courier Time Server
  - — Backup Courier Time Server
- DTS Application Programming Interface
- Time Provider Interface (TPI)
- 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.

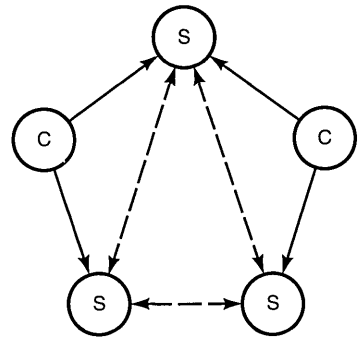
Time Clerk. The Time Clerk is the client side of DTS. It runs on a client machine, such as a workstation, and keeps the machine's local time synchronized by asking Time Servers for the correct time and adjusting the local time accordingly.

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 3-14 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 3–14. DTS Time Clerks and Servers



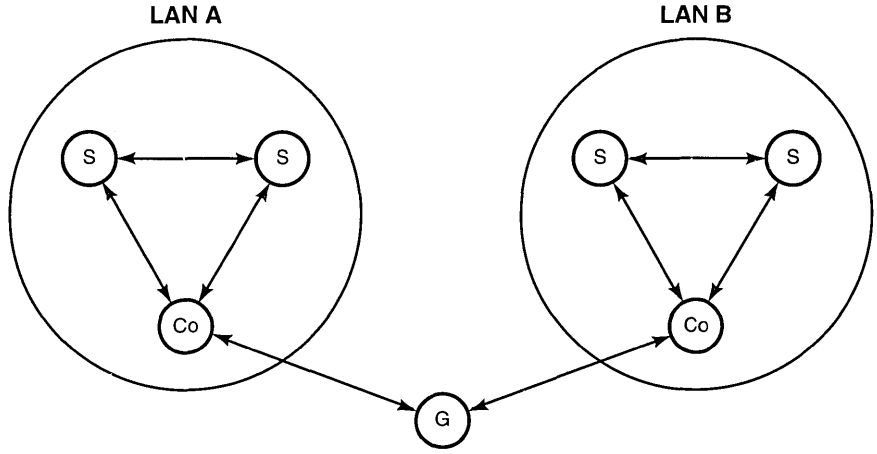
Time Servers. A Time Server is a node that is designated to answer queries about the time. The number of Time Servers in a DCE cell is configurable; three per LAN is a typical number. Time Clerks query these Time Servers for the time, and the Time Servers query one another, computing the new system time 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 3-15 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) (that is, a Time Server outside of either LAN) for the current time.

Figure 3–15. Local, Courier, and Global Time Servers



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—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.

3.4.2 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.

3.4.3 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 or 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.

3.4.4 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.

3.4.5 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 *OSF DCE Administration Guide* for additional information.

3.4.6 Additional Information on DTS

For additional information on the DCE Distributed Time Service, see the following:

- The DTS chapters of the *OSF DCE Application Development Guide* and the *OSF DCE Administration Guide*
- The **(3dts)** reference pages of the *OSF DCE Application Development Reference*
- The **(8dts)** reference pages of the *OSF DCE Administration Reference*

3.5 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.

3.5.1 What Is the 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, and the Login Facility.

- The Authentication Service

  The Authentication 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.

- The 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 DCE Authentication Service and the DCE 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.

- The Registry Service

  The DCE 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 Access Control List Facility

  DCE Access Control Lists (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 the **acl_edit** command allows users to modify ACLs associated with resources they own.

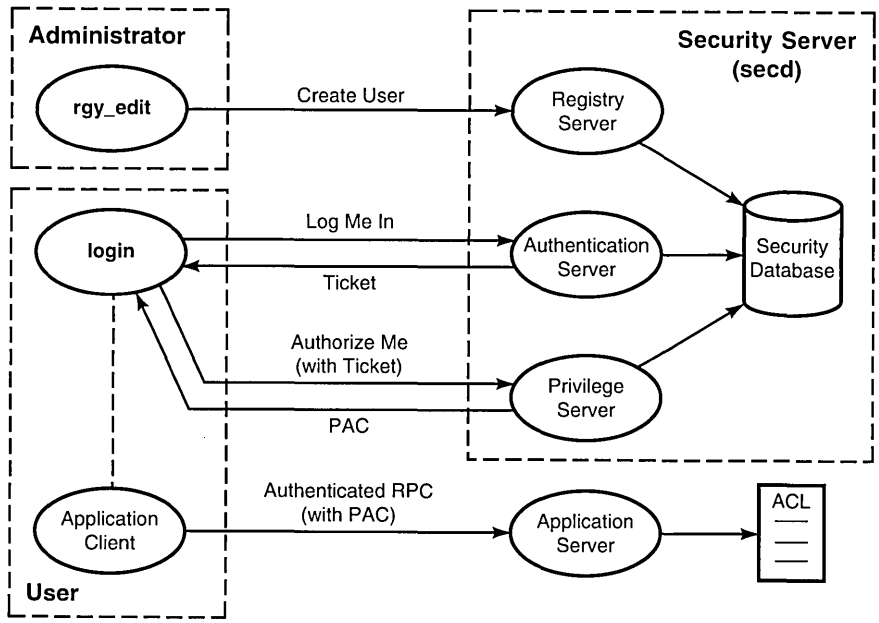
- The Login Facility

  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 the Distributed File Service or other applications.

3.5.2 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 3-16 shows this process. The presentation in this section is a simplified view of the transactions that actually take place.

Figure 3–16. 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 the **rgy_edit** command, 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 in 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 a PAC (Privilege Attribute Certificate). The PAC contains authorization information specific to the user, such as which groups the user belongs to. PACs 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 PAC. The server inspects the user's authorization credentials and the Access Control List 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 PAC 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 used in DCE Release 1.0 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 *OSF DCE Application Development Guide*.)

3.5.3 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 the **acl_edit** program. Finally, a user notices the Security Service in action when he or she is denied unauthorized access to resources.

3.5.4 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, Login, 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 Version 1.0, two types of authentication service are supported—secret key authentication, which is based on Kerberos (see Section 3.5.6), 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.

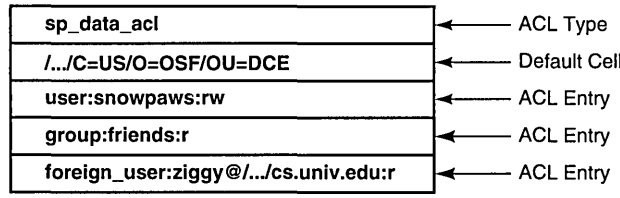
The **authorization service** chosen by the client can be either uncertified or certified. In uncertified authorization, the authorization information sent to a server consists only of the client's name. In certified authorization, the authorization information consists of the UUIDs of the client's name and groups. The certified authorization information is in the form of a Privilege Attributes Certificate (PAC), 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 Access Control Lists and determine whether a client should be given the access it is requesting.

Access Control Lists. 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 Access Control List 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 3-17 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=OSF/OU=DCE**. The rest of the ACL consists of ACL entries.

Figure 3–17. DCE ACL Example



ACL entries can be of several types. The 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 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=OSF/OU=DCE** read and write permission to the object, the members of the **friends** group in the **/.../C=US/O=OSF/OU=DCE** cell read permission to the object, and the principal **ziggy** in the foreign cell **/.../cs.univ.edu** read permission.

3.5.5 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.)

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 **rgy_edit**, and controlling replication of security data.

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.

3.5.6 DCE Security and Kerberos

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.

3.5.7 Additional Information on DCE Security

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

- The Security chapters of the *OSF DCE User's Guide and Reference*, the *OSF DCE Application Development Guide*, and the *OSF DCE Administration Guide*

- The **(3sec)** reference pages of the *OSF DCE Application Development Reference*
- The **(8sec)** reference pages of the *OSF DCE Administration Reference*

3.6 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. The 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's Distributed File Service (DFS) is a distributed client/server application, built on the underlying DCE services. It takes full advantage of both the lower-level DCE services (such as RPC, Security, and Directory) and the distributed computing system itself. The following subsections describe DFS, the configuration of its components, and various user perspectives on DFS.

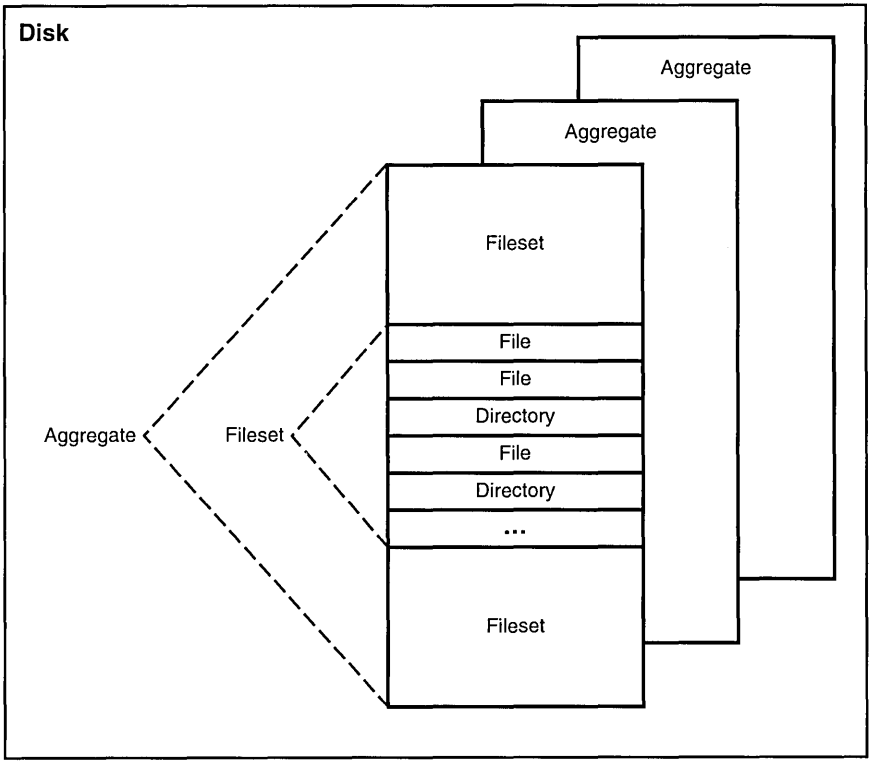
3.6.1 What is DFS?

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.

3.6.1.1 DFS Data Organization

DFS data is organized at three levels (see Figure 3-18).

Figure 3–18. Files, Directories, Filesets, and Aggregates



The three levels of DFS data are, from smallest to largest:

- Files and Directories

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

- Filesets

  The **fileset** is the unit of administration. A fileset is a subtree of files and directories, 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 on the same fileset.

- Aggregates

  The **aggregate** is the unit of disk storage, similar to a disk partition. It is also the unit of fileset exporting. It can contain one or more filesets.

3.6.1.2 DFS Components

DFS is implemented by several components; this section describes each of them briefly, beginning with the software that runs on DCE client machines (the Cache Manager), then the software that runs on DCE File Server machines (the File Exporter, DCE LFS, and Token Manager), and finally the administrative servers, which may also run on DFS File Server machines (the Fileset Server, Basic OverSeer Server, Replication Server, Update Server, Scout, Backup Server, and Fileset Location Server).

Cache Manager. The Cache Manager is the client side of DFS, which runs on any machine acting as a DFS client. It takes a user's file system request and looks in a cache to see if a copy of the data is already on the local system. If not, the Cache Manager sends a request to the File Server machine for the data and caches it locally. Since files are cached on the client, a local copy of the 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 always had to go to the server each time it needed to access the file.

File Exporter. The File Exporter is the server side of DFS. It runs on a DFS File Server machine, where it handles requests from remote 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 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, which may be concurrently accessing the same file, and returns the requested information to the client.

DCE Local File System. The 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, and LFS is analogous to a UNIX file system. However, LFS is more powerful than most UNIX local file systems—it includes features that result in greater capabilities than a distributed file service based on a traditional UNIX file system. These

capabilities include the ability to use more flexible authorization in the form of DCE Access Control Lists (ACLs); the ability to replicate, back up, and even move different parts of the file system without interruption in service; and fast recovery after a crash, made possible through logging (in contrast to UNIX file systems, which must execute the time-consuming **fsck** command). DCE LFS includes support for DCE cells; for example, the owner of a file or name in an Access Control List can be a name in a foreign cell.

A UNIX File System (UFS) can be used as the File Server machine's physical file system as an alternative to LFS. DFS can export UFS, issue synchronization tokens for files in UFS, and perform fileset operations such as dump and restore on UFS. However, there is only one fileset per UFS partition, which results in large filesets that are not as convenient to move. Also, UFS filesets cannot be replicated. So a File Server machine using LFS will have more functionality than a File Server machine using UFS, although UFS systems can be supported in DFS.

Token Manager. The Token Manager runs on the File Server machine and synchronizes 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, then the conflicting token is revoked before the requested operation is allowed to proceed.

Fileset Server. The Fileset Server allows administrators to create, delete, move, and perform other operations on filesets. For example, an administrator can use the Fileset Server 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, then an entire partition is treated as a single fileset and 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. It maintains information about those 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 a copy of a fileset on a second File Server machine. The Replication Server will update the replica at a specified interval (every 30 minutes, for example). This means that even if the file's master File Server machine goes down, a copy of the file is still available online through the secondary copy on the alternate File Server machine.

Update Server. The Update Server provides the ability to distribute binary files or administration information to nodes in the DFS system. The Update Server consists of an **upclient** and an **upserver**. The **upclient** software runs on a machine that needs to receive new versions of the binary files or administration information. The **upserver** program runs on a master machine and automatically propagates any changes to binaries or administration information to the machines running the **upclient** software.

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 DFS system.

Backup Server. The Backup Server is a facility for backing up File Server data. It maintains backup records in the Backup Database. It schedules file system backups, and has the ability to make incremental dumps. The unit for backup is the fileset.

Fileset Location Server. The Fileset Location Server is a replicated directory service, which keeps track of which filesets can be found on which File Server machines. It provides lookup service analogous to the service CDS provides, except the Fileset Location Server is specialized for DFS. It supports fileset location transparency, since a fileset can be accessed simply by knowing its name rather than having to know where it resides. As a result, a fileset can be moved and its location can be updated in the Fileset Location Database without users and applications having to know about the move.

Some DFS components run in the host machine's kernel. These are the Cache Manager and Token Management Layer on DFS client machines; the File Exporter, Token Manager, and DCE Local File System on File Server machines; and parts of the Fileset Location Server.

3.6.1.3 Features of DCE DFS

The DCE Distributed File Service has the following features:

- 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(s) on which the file is located in order 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 the DCE system, including from foreign cells.

- 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, an administrator can move a fileset from one File Server machine to another for load balancing, for example, without disturbing users.

- 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 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 accesses to the file can then be made to the copy on the client machine rather than to the File Server machine.

- Availability

  DFS makes its service and data highly available in several ways. One way is through replication—a copy of a file can be stored on more than one File Server machine. This way, if the file's main File Server machine is down, a copy of the file may still be available (although possibly not up-to-date) on another File Server machine. This type of replication is especially useful for files that are accessed by many users and change infrequently (for example, binary files).

Another way DFS achieves high availability is through caching. Copies of files are cached on DFS clients, so even if a client is temporarily disconnected from the network, it may be possible for a user to access a file since a copy may 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 files can be done without making the files unavailable to users.

The physical file system portion of DFS, called the DCE Local File System, is designed for fast recovery (yielding higher availability) after failures. This is possible because DCE LFS is a log-based file system; that is, 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.

- 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 the network communications and movement, synchronization, and storage of shared data.

- 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 File Exporter and backup processes, are monitored and maintained automatically by the Basic OverSeer Server, resulting in less work for a DFS administrator and a more scalable system. Because of the high performance mentioned previously, DFS has a high client-to-server ratio. This leads to a scalable system—clients can be added with low impact on other clients and the rest of the system. Finally, DFS includes tools such as the Backup Server and Update Server that automate time-consuming administrative tasks.

- Integration

  DFS is fully integrated with other DCE components, including RPC, Security, Directory Service, and Threads.

- Interoperation

  DFS interoperates with other file systems; for example, a UFS file system can be exported using DFS.

- Standards

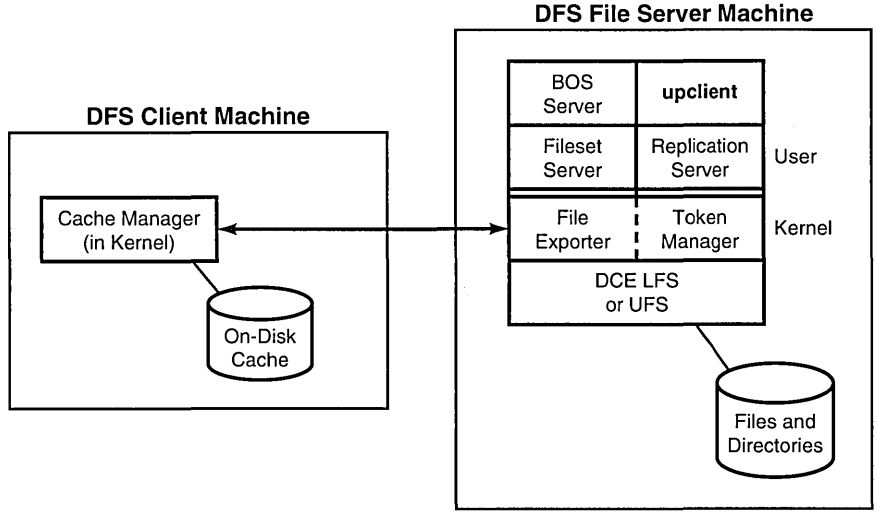
  DFS maintains POSIX single-site read/write semantics. The DCE Local File System is POSIX 1003.1 conformant.

3.6.2 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 3-19).

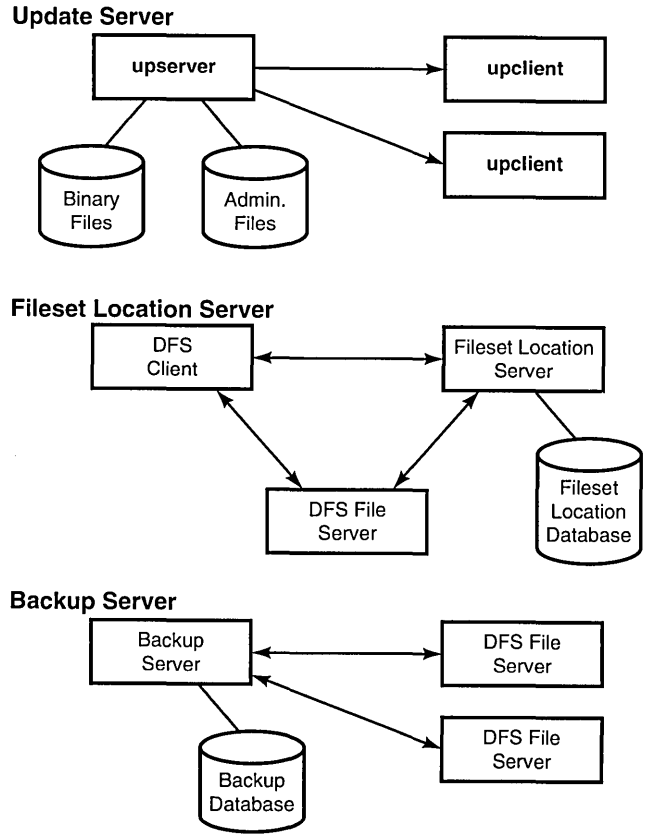
Figure 3–19. DFS Client and File Server Machines



Several processes run on DFS File Server machines: the File Exporter (which includes a Token Manager), the Basic OverSeer 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, either DCE LFS or UFS.

Finally, some processes must run on a machine that contains the database they access (usually the DFS File Server machine). See Figure 3-20.

Figure 3–20. Other DFS Servers



These processes are the server side of the Update Server (which runs both on machines containing master copies of configuration files and machines containing master copies of binary files), the Fileset Location Server (which runs on the machine where the Fileset Location Database is located), and the Backup Server (co-located with the Backup Database).

3.6.3 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, and some in another cell—to a user, the DCE Distributed File Service presents one large, worldwide file system. Users will 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 and can run commands for information about quotas. There are also commands for determining the location of a file, and other information that is specific to a distributed file system.

3.6.4 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 the Distributed File Service, programmers can write distributed applications without the use of RPC and the client/server model, if the DFS data sharing model is appropriate to the application.

3.6.5 DFS Administration

Administration of DFS is a significant task, since there are several processes that implement DFS that 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.

3.6.6 Additional Information on DFS

For additional information on the DCE Distributed File Service, see the following:

- The DFS chapters and **(1dfs)** reference pages of the *OSF DCE User's Guide and Reference*
- The DFS chapters of the *OSF DCE Application Development Guide* and the *OSF DCE Administration Guide*
- The **(2dfs)** and **(3dfs)** reference pages of the *OSF DCE Application Development Reference*
- The **(1dfs)**, **(4dfs)**, and **(8dfs)** reference pages of the *OSF DCE Administration Reference*

3.7 DCE Diskless Support Service

The DCE Diskless Support Service enables nodes without disks to participate in a DCE environment. A diskless node has no disk on which to store a local file system. Diskless support means providing facilities over the network to replace the functionality that the local disk traditionally supports. The four areas that must be dealt with are as follows:

- Booting a kernel
- Obtaining configuration information
- Remote file system support
- Remote swapping support

When starting up, a traditional machine looks for a binary image of its operating system in a well-known file on its local disk (such as **/vmunix** on many UNIX machines). A diskless system, since it has no local disk on which to store its kernel, must be able to boot by some other method. In DCE, this is done by obtaining a copy of the kernel over the network.

Once a diskless client is booted, there is still some information it needs before it can be fully operational. Since it still cannot read that information off the root file system (it does not have one yet), it has to get it from another source. The diskless node obtains this information from a Diskless Configuration (DLC) Server.

One of the main functions of a local disk is the permanent storage of user and system files. A diskless system needs a place to store its files, since it cannot store them locally. DCE provides the ability for diskless machines to use a remote file server to store and retrieve their files.

Another typical use of local disks on traditional machines is for swapping. When a process is blocked and the CPU needs more space in main memory to run the next process, the disk can be used to temporarily store the blocked process. When it is time for the blocked process to run again, it is brought back into main memory from the disk. A diskless node needs another way to swap processes in and out of its memory, since it cannot use a disk. In DCE, this is done by using a remote swap server; that is, by using the disk on another machine on the network for the diskless machine's swap space. (Depending on the diskless machine's operating system, this facility may be used for paging as well as swapping.) For diskless machines using DFS, paging can be done to files instead of through the Swap Server.

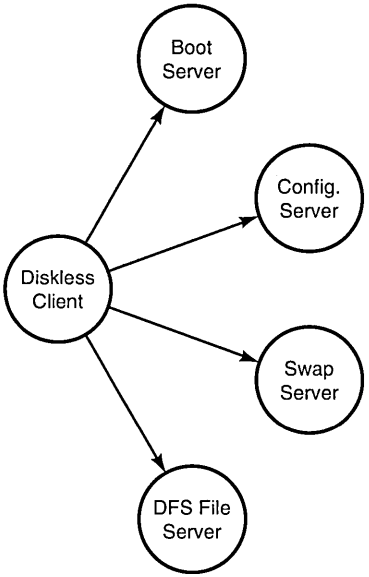
3.7.1 What Is DCE Diskless Support Service?

DCE diskless configurations involve four types of machines (see Figure 3-21):

- Diskless Client (BOOTP and TFTP clients)
- Boot Server (**bootpd** and **tftpd**)
- Configuration Server
- Swap Server (optional)
- DFS File Server

Some of these servers can be co-located; for example, the Boot Server and Configuration Server can run on a Distributed File Server machine. (The Boot Server and Configuration Server must run on the same machine.)

Figure 3–21. Diskless Client and Related Servers



3.7.2 Booting Support for Diskless Operation

Booting support consists of code that resides in the diskless client machine's Read-Only Memory (ROM), since that is the only place on a diskless node that is available for permanent storage. The code contains the client sides of the BOOTP and Trivial File Transfer Protocol (TFTP) protocols. Since DCE is limited to offering software, not hardware, the actual supplied code runs in user mode and demonstrates the ability to get a kernel over the network using these two protocols.

During the boot process, the diskless client uses BOOTP to obtain its own network address and the address of its Boot Server. It then uses TFTP to obtain the kernel file to be booted.

3.7.3 The Diskless Configuration Process

A diskless node needs configuration information before the root file system is available; this is obtained from the Diskless Configuration Server. This server supplies the diskless node with information about the following:

- The Client's Swap Server (if needed)
- The Client's root file system location and server
- DFS Cache Manager configuration parameters

Using these configuration values, the diskless node starts the DFS Cache Manager and mounts the appropriate root file system from the DFS File Server machine.

3.7.4 File Service Support for Diskless Operation

DCE Diskless Support Service provides the ability for a diskless machine to use a remote file service. This facility is based on the DCE Distributed File Service (see Section 3.6), and it has several aspects:

- Diskless Cache Manager
- Machine-Specific Files
- Device Files

When DFS is used on a machine with a local disk, the DFS Cache Manager (the client side of DFS) uses storage on the disk to keep a cache of recently accessed files. This greatly reduces network traffic to the File Server machine, and increases performance. On a diskless machine, no local disk is available, so the Cache Manager uses an in-memory cache instead.

Since DCE supports interoperation among heterogeneous systems, it is possible that a File Server machine may have clients with different kinds of systems. A mechanism is needed to distinguish between, for example, binary files intended for one platform, and binaries for a second platform, so that when a diskless client needs to execute the binary file, it gets the right version for its system. DFS provides a mechanism for making this distinction in the file system.

The machine-specific file mechanism consists of two special filenames:

- **@host**
- **@sys**

The **@host** filename is replaced with the hostname of the client. This special filename is used for files that are specific to the individual client machine; for example, an **/etc/rc** file or equivalent.

The **@sys** filename is replaced with the client's system name. For example, the diskless server may have copies of executable files for various platforms—the **@sys** filename is replaced with the specific client's platform, so the client can access the appropriate executables for its operating system/hardware configuration.

Finally, some of the files in a DCE file system are actually devices (that is, **/dev** files). These are interpreted as devices on the client machine, not as devices on the server machine.

3.7.5 Swapping Support for Diskless Operation

DCE provides diskless swapping support for systems that swap or page to devices; systems that page to files do not need this support. The diskless swapping support comprises the following components:

- Swap Server

  On the Swap Server machine, the diskless swap server daemon, **dswd**, runs in user mode. It maintains a database of information about its clients, swap files, and swap devices, and accepts administrative requests to modify that information. It also accepts requests from users to access swap space, which it keeps on disk.

- Client Swap Driver

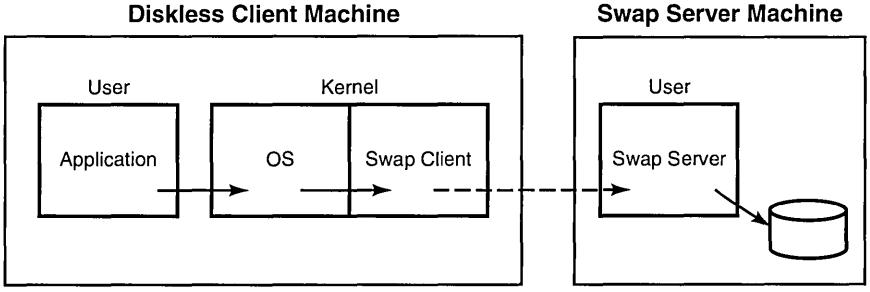
  On the diskless node, the Swap Client runs in kernel mode. It looks like a device driver to the diskless node's operating system. It takes swap requests from the client's kernel and forwards them across the network to the Swap Server.

- Swap Administration Program

  The diskless swap administration program, **dsw_adm**, allows an administrator to control the Swap Server remotely. This program makes requests to the Swap Server to modify the Swap Server's database. Requests modify client information and information about files and devices to be used for swapping.

Figure 3-22 shows the interaction among an application on the diskless machine, its operating system and swap driver, and the Swap Server and swap store on the Swap Server machine.

Figure 3–22. Swap Process from Application to Server



3.7.6 Additional Information on DCE Diskless Support

For additional information on the DCE Diskless Support Service, see the following:

- The Diskless module of the *OSF DCE Administration Guide*
- The **(8dskl)** reference pages of the *OSF DCE Administration Reference*
- The Diskless sections of the *OSF DCE Porting and Testing Guide*

3.8 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. For a more extensive application example, which uses many more DCE services and facilities, see the **timop** example in Part 1 of the *OSF DCE Application Development Guide*.

3.8.1 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.

3.8.1.1 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 labelling the application's interface. It also creates a template for an IDL file. The following command:

   uuidgen -i > greet.idl

   creates the file **greet.idl**.

It 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 comprising 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 the Makefile in Section 3.8.1.3 for the complete command.) Three new files are created automatically as a result of this command:

   - **greet.h**
   - **greet_cstub.o**
   - **greet_sstub.o**

5. Write the client application code **greet_client.c**.

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

   - The Client code
   - The Server initialization code
   - 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 (RPCD).
     */
    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 the Cell Directory Service. 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
```

3.8.1.2 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.

- 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.

- 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
```

- Running the **greet** Application

  To run the **greet** application, a user types 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.

3.8.1.3 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
```

3.8.2 The greet Application: An Implementation Using DCE DFS

This section describes an implementation of the **greet** application using the DCE Distributed File Service. 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 using 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.

(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**.

- 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"
```

- 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: %s0, s);
        unlink(S_GREET_FILE);
}
```

- 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: %s0, 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 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 the OSF Distributed Computing Environment 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, the Cell Directory Service 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.

4.1 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 4–1. DCE Component Integration

	Threads	RPC	CDS	DTS	Security	GDS	DFS	Diskless
Threads	NA							
RPC	X	NA	X		X			
CDS	X	X	NA	X	X	X		
DTS	X	X	X	NA	X			
Security	X	X	X	X	NA			
GDS						NA		
DFS	X	X	X	X	X		NA	
Diskless	X	X	X				X	NA

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 the Cell Directory Service, while CDS uses the Security Service to control access to its information. The implications of these mutual dependencies are discussed in Section 4.3.

4.2 Integration by Technology Component

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

- DCE Threads Integration

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

- 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. GDS can be used indirectly by RPC. If an RPC server is located in a foreign cell that is registered in the X.500 namespace, then GDS is accessed via CDS to find the given RPC server.

  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 time-out 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.

- 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 GDS (via the GDA) to find foreign cells

registered in GDS. And finally, CDS uses DCE Security's Access Control Lists and authenticated RPC to ensure authorized access to directory data and administrative functions.

- 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. GDS may be used indirectly if a Global Time Server is registered in a foreign cell that is registered in the X.500 namespace. DTS uses the DCE Security Service to authenticate its interactions.

- DCE Security Service Integration

  The DCE Security Server, like all DCE RPC-based applications, uses DCE Threads. The Security Server communicates with its clients using DCE RPC. CDS is used to find Security Servers. GDS may be used indirectly in accessing a Security Server that is in a foreign cell registered in the X.500 namespace.

  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 the Distributed Time Service. The Security Service does not use DTS timestamps in this version of DCE.

- DCE GDS Integration

  The GDS server does not use DCE Threads in DCE Release 1.0; instead, it uses multiple processes to handle multiple requests. Since GDS is based on the X.500 standard, which is specified to run over ISO protocols, GDS does not use DCE RPC.

  GDS does not use CDS; since GDS is at a higher level in the global namespace hierarchy, CDS refers to GDS but not the other way around. GDS has a separate security mechanism and ACLs from the DCE Security Service. Again, this is in order for GDS to comply to the international directory service standard.

- 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 may use GDS indirectly (via CDS) to locate Fileset Location Servers in foreign cells registered in the X.500 namespace. DFS uses authenticated RPC and DCE ACLs to protect its resources. DFS relies on DTS to maintain clock synchronization in the network.

- DCE Diskless Support Service Integration

  The Swap Server component of the Diskless Support Service uses DCE Threads for concurrency. The Diskless Support Service uses RPC for its interactions with CDS and between the Swap Client and Server. The Diskless Cache Manager also uses RPC to communicate with the DFS File Exporter. It does not use RPC for booting, however, since booting requires very small, simple network services. Diskless Support uses CDS to find its configuration information. Diskless Support Service includes instructions on how to modify the diskless node's kernel to use the DFS Cache Manager (DFS client) to mount the client's root file system.

  For diskless operation to be secure, it would require hardware support, which is outside the scope of DCE.

4.3 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.

- 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 first module of the *OSF DCE Administration Guide*.

- 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, since 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. GDS can be ported independently, since it has no dependencies on other DCE components. For information on porting DCE technologies, see the *OSF DCE Porting and Testing Guide*.

- 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 first module of the *OSF DCE Administration Guide*.

- 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 introductory chapter and the Threads chapters of the *OSF DCE Application Development Guide* for information about programming with Threads.

Appendix A

Overview of DCE Documentation

This appendix describes the documentation set supplied with the OSF DCE offering and suggests reading paths for different audiences. Except where noted, the manuals comprising the DCE documentation set are made available through Prentice-Hall.

A.1 DCE Documentation

The DCE documentation is organized under the following titles:

- *Introduction to OSF DCE*
- *OSF DCE User's Guide and Reference*
- *OSF DCE Administration Guide*
- *OSF DCE Administration Reference*
- *OSF DCE Application Development Guide*
- *OSF DCE Application Development Reference*
- *OSF DCE Porting and Testing Guide*

- *OSF DCE Technical Supplement*
- *OSF DCE Release Notes*
- *Application Environment Specification/Distributed Computing*

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

A.1.1 *Introduction to OSF DCE*

This manual is the *Introduction to OSF DCE*. It provides an overview of DCE, and serves as an introduction to the rest of the DCE documentation. It also contains the Glossary of terms used in DCE documentation.

A.1.2 *OSF DCE User's Guide and Reference*

This guide presents task and reference material for the DCE end user. Most of what the DCE user sees is related to the DCE Distributed File Service and the Access Control List facility of the DCE Security Service. Since much of DCE is software for supporting the development and execution of distributed applications, much of the DCE documentation is targeted for application programmers and administrators rather than end users. However, some parts of DCE are actually distributed applications (for example, the DCE Distributed File Service), and are therefore seen at the end-user level.

A.1.3 *OSF DCE Administration Guide*

This guide, which consists of seven modules, provides conceptual and task-oriented information for a DCE administrator. The first module is an overview, which describes administering DCE as a whole, including planning and configuring information. Subsequent modules are devoted to the management of specific components—RPC, Directory Service, Distributed Time Service, Security Service, and Distributed File Service. This guide is available through DCE licensees and OSF Educational Services.

A.1.4 *OSF DCE Administration Reference*

This manual provides reference material for commands needed by the DCE administrator. It is divided into technology component sections.

A.1.5 *OSF DCE Application Development Guide*

This guide is targeted for the distributed application programmer. It provides conceptual and task-oriented information for developing an application using DCE. The first chapter describes programming with DCE in general, providing a typical example. Subsequent chapters explain the use of application programming interfaces for each of the relevant technology components.

The DCE application programmer typically uses the RPC facility, which in turn uses other components. The bulk of the guide therefore describes the use of RPC. Other components are also described, since they are useful for more specialized applications.

A.1.6 *OSF DCE Application Development Reference*

This manual provides reference material for the DCE programming interfaces. It also has command references for a few commands needed by the DCE programmer; in particular, those used with the RPC component.

A.1.7 *OSF DCE Porting and Testing Guide*

This guide describes the DCE code and documentation source trees, issues that arise when porting the different components to new platforms, and how to test software that has been ported or rebuilt. It also contains information for improving the performance of various components. This manual is available through DCE licensees and OSF Educational Services.

A.1.8 *OSF DCE Release Notes*

The *OSF DCE Release Notes* describe a given version of DCE software from OSF. They include information on building the code and documentation, and known defects and restrictions. This manual is available through DCE licensees and OSF Educational Services.

A.1.9 *Application Environment Specification/Distributed Computing*

The *Application Environment Specification/Distributed Computing (AES/DC)* specifies the interfaces, services, and protocols that comprise the core, stable components of the DCE offering. It is intended for systems vendors who wish to implement a conformant DCE offering, or for application writers who wish to write programs that are portable to any conformant DCE system.

A.1.10 *OSF DCE Technical Supplement*

This manual describes internal DCE interfaces and protocols. It also contains architectural specifications to provide the interested reader with conceptual information about the DCE components. This manual is of interest primarily to developers who are extending DCE, and is available only to DCE licensees; it may not be redistributed.

A.2 Reading Paths

This section suggests reading paths through the DCE documentation for various audiences, including the following:

- People interested in a high-level overview of DCE
- End users
- Application programmers

- System administrators
- DCE developers
- DCE implementors

Note that the *OSF DCE Release Notes* describe a given version of DCE, and are therefore potentially of interest to all DCE audiences, in addition to the documents listed for each audience below.

A.2.1 High-Level Overview of DCE

For a high-level overview of DCE, including its architecture, components and potential use, read this manual, the *Introduction to OSF DCE*.

A.2.2 End Users

The curious user might begin by reading the High-Level Overview path to get an idea of DCE as a whole. The *OSF DCE User's Guide and Reference* provides information on the specific parts of DCE that are visible to the user, in particular the DCE Distributed File Service, viewing the DCE Directory Service namespace, and using Access Control Lists to protect users' resources.

A.2.3 Application Programmers

Application programmers may wish to begin with the High-Level Overview in order to understand the system as a whole. Alternatively, they can begin directly with information pertaining to programming DCE, starting with the *OSF DCE Application Development Guide*. For detailed information on a specific function or command, see the *OSF DCE Application Development Reference*.

A.2.3.1 System-Specific Applications

Applications for a specific DCE implementation, such as a reference implementation provided by OSF, or a DCE implementation provided by a vendor for a specific platform, can be written by following the documentation mentioned in the previous section.

A.2.3.2 Portable Applications

Applications written to be portable across multiple DCE implementations, such as applications written by Independent Software Vendors, should be limited to using only the interfaces contained in the *Application Environment Specification/Distributed Computing (AES/DC)*. Those interfaces are a subset of the interfaces contained in the DCE documentation, and the *AES/DC* interfaces are the ones that every DCE-conformant implementation supports.

A.2.4 System Administrators

We divide the audience of system administrators into two categories: those who are planning, configuring, and installing DCE; and those who are responsible for maintaining DCE once it is up and running.

People in the first category should begin with the *Introduction to OSF DCE* to gain an understanding of the whole system. Next, the first module of the *OSF DCE Administration Guide* provides insight into issues and conventions pertaining to DCE as a whole system, and provides guidance for planning and configuring a DCE system. Administrators may then want to read the *OSF DCE Release Notes* to learn how to build and install the various components of a DCE system. Finally, specific sections on a given component to be installed (for example, the Directory Service or Time Service) should be reviewed in the *OSF DCE Administration Guide*, and for even more detailed information, the *OSF DCE Administration Reference*.

Administrators in the second category, DCE system maintainers, should refer first to the *OSF DCE Administration Guide*, and then the *OSF DCE Administration Reference*, for information on the particular DCE component they are administering.

A.2.5 DCE Developers

Some of the audience for DCE documentation are system developers, such as system vendors who are taking the DCE source code and extending it or modifying it to suit the specific requirements of their customers. This audience should begin by reading the *OSF DCE Release Notes* to determine the state of DCE software in the specific release they are working with. They may also want to read the *Introduction to OSF DCE* for an overview of the system.

The *OSF DCE Application Development Guide* and *OSF DCE Application Development Reference* may be helpful in understanding the use of the other DCE components by the component being developed. Finally, the *OSF DCE Technical Supplement* has been included in the DCE documentation set particularly for the DCE developer. It contains information on DCE internals, architectures, and concepts. This information is not needed by most DCE end users or programmers, but may be very helpful to a DCE developer. The *OSF DCE Porting and Testing Guide* also contains useful information for this audience.

A.2.6 DCE Implementors

Finally, one expected audience for DCE documentation is people who are developing their own implementation of part or all of DCE. They should begin by reading the *Introduction to OSF DCE*. Then they should consult the appropriate sections of the *Application Environment Specification/Distributed Computing* to determine which interfaces need to be implemented to have a conformant DCE implementation. If the implementor is also a DCE licensee, the documentation listed previously for the DCE developer will probably also be helpful in the implementation project.

Appendix B

DCE Documentation Outline

This appendix gives an outline of the DCE documentation set contents. The purpose is to give readers looking for information on a particular subject an idea of where to look. For example, readers interested in information on DCE Threads can see which sections of the documentation contain relevant information, noting that there is no applicable (NA) information for Threads administration.

Table B–1. DCE Documentation Outline

Introduction to OSF DCE	
Component or Subject	**Chapter or Part**
Overview	Chapter 1
Configuration	Chapter 2
Technology Components	Chapter 3
Integration	Chapter 4
Glossary	

OSF DCE User's Guide and Reference	
Component or Subject	**Chapter or Part**
Threads	NA
RPC	NA
CDS	Guide
GDS	NA
GDA	NA
XDS	NA
DTS	NA
Security	Guide
Security	Reference (1sec, 1)
DFS	Guide
Diskless	NA

OSF DCE Application Development Guide	
Component or Subject	**Chapter or Part**
All Components	Part 1
Threads	Part 2
RPC - Using RPC	Part 3A
RPC - Language Syntax	Part 3B
RPC - Supplement	Part 3C
CDS	NA
GD	NA
GDA	NA
XDS	Part 4
XOM	Part 5
DTS	Part 6
Security	Part 7
DFS	Part 8
Diskless	NA

OSF DCE Application Development Reference	
Component or Subject	**Chapter or Part**
Threads	(**3thr**)
RPC	(**1rpc**, **3rpc**, **7rpc**)
CDS	NA
GDS	NA
GDA	NA
XDS	(**3xds**, **4xds**)
XOM	(**3xom**, **4xom**)
DTS	(**3dts**)
Security	(**3sec**)
DFS	(**2dfs**, **3dfs**)
Diskless	NA

OSF DCE Administration Guide	
Component or Subject	**Chapter or Part**
All Components	Module 1
Threads	NA
RPC	Module 2
CDS	Module 3
GDS	Module 4
GDA	in CDS Module
XDS	NA
DTS	Module 5
Security	Module 6
DFS	Module 7
Diskless	Module 8

OSF DCE Administration Reference	
Component or Subject	**Chapter or Part**
Threads	NA
RPC	Part 1 (**8rpc**)
CDS	Part 2 (**8cds**)
GDS	Part 3 (**8gds**)
GDA	in CDS Part
XDS	NA
DTS	Part 4 (**8dts**)
Security	Part 5 (**8sec**)
DFS	Part 6 (**1dfs**, **4dfs**, **8dfs**)
Diskless	Part 7 (**8dskl**)

OSF DCE Porting and Testing Guide	
Component or Subject	**Chapter or Part**
Building	Chapter 1
Threads	Chapter 2
RPC	Chapter 3
CDS	Chapter 4
GDS	Chapter 5
GDA	in CDS Chapter
XDS	in GDS Chapter
DTS	Chapter 6
Security	Chapter 7
DFS	Chapter 8
Diskless	Chapter 9
System Testing	Chapter 10

OSF DCE Technical Supplement	
(available to DCE licensees only)	
Component or Subject	**Chapter or Part**
Introduction	Part 1
Threads	Part 2
RPC	Part 3
CDS	Part 4
GDS	Part 5
GDA	Part 6
XDS	in GDS Part
DTS	Part 7
Security	Part 8
DFS	Part 9
LFS	Part 10
Diskless	Part 11

OSF DCE Release Notes	
Component or Subject	**Chapter or Part**
Overview Release 1.0	Chapter 1
Building & Installing	Chapter 2
Restrictions & Defects	Chapter 3
Building Documentation	Chapter 4

Appendix C

List of Acronyms and Abbreviations

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

Table C–1. DCE Acronyms and Abbreviations

Acronym/Abbreviation	Definition
ACF	Attribute Configuration File
ACL	Access Control List
ACSE	Association Control Service Element
AES	Application Environment Specification
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
AVA	Attribute Value Assertion
BER	Basic Encoding Rules
BOS	Basic OverSeer Server
C	Country

Acronym/Abbreviation	Definition
C-ISAM	C-language Indexed Sequential Access Method
CCITT	International Telegraph & Telephone Consultative Committee
CDS	Cell Directory Service
CDSPI	Cell Directory Service Portable Interface
CPU	Central Processing Unit
DAP	Directory Access Protocol
DB	Database
DCE	Distributed Computing Environment
DFS	Distributed File Service
DIB	Directory Information Base
DIT	Directory Information Tree
DLC	Diskless Configuration Service
DN	Distinguished Name
DNS	Domain Name Service
DSA	Directory System Agent
DSP	Directory System Protocol
DTS	Distributed Time Service
DUA	Directory User Agent
FIFO	First In, First Out
GDA	Global Directory Agent
GDS	Global Directory Service
IDL	Interface Definition Language
IP	Internet Protocol
ISO	International Organization for Standardization
LAN	Local Area Network
LFS	Local File System
LRU	Least Recently Used
MAVROS	Not an acronym
MS-DOS	Microsoft Disk Operating System

Acronym/Abbreviation	Definition
NA	Not Applicable
NetBIOS	Network Version of Basic Input/Output System
NSI	Name Service Independent
NTP	Network Time Protocol
O	Organization
OS	Operating System
OS/2	Operating System/2
OSF	Open Software Foundation
OSI	Open Systems Interconnection
OSS	OSI Session Service
OU	Organizational Unit
RDN	Relative Distinguished Name
ROM	Read-Only Memory
ROS	Remote Operation Service
ROSE	Remote Operation Service Elements
RPC	Remote Procedure Call
RR	Resource Record (DNS)
RR	Round Robin (scheduling)
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
UUID	Universal Unique Identifier
VFS	Virtual File System
WAN	Wide Area Network
XOM	X/Open Object Management

Acronym/Abbreviation	Definition
XDS	X/Open Directory Service
XTI	X/Open Transport Interface

Glossary Usage

The Glossary defines terms used in this manual and the rest of the DCE documentation set. Each term is defined for the audience of the manual in which it appears. In some cases, a given term has a different meaning when used in the context of different technology components. This 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.

Glossary

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 access control list 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. 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 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 access control list 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 point

The point at which an Abstract Service is obtained. (A connection between a DUA and a DSA.)

access right

See **permission**.

accessible

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

account

Data in the Registry database that allows a principal to log in. An account is indistinguishable from a principal identifier and is the registry object that represents a principal.

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 non-null 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

GDS: A name for a (directory) object, provided by the use of one or more alias entries in the DIT.

alias entry

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

alias, alias name

GDS: A name for a directory object, which is provided by the use of one or more alias entries in the DIT.

aliased object

Object to which an alias entry refers.

aliasing

RPC: Aliasing 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 on to the directory service without giving a name and password.

API

See **Application Programming Interface**.

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: An **.acf** file. An optional companion to an interface definition file (an **.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 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.

attribute value

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

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 also **attribute syntax, syntax.**

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 three services provided by DCE Security: the Authentication Service authenticates principals according to a specified authentication protocol. The other Security services are the Privilege Service and the Registry Service.

authentication service

See **authentication protocol.**

authentication surrogate

A type of principal represented by an entry in a cell's Registry database that specifies the same secret key as a corresponding entry in another cell's Registry database. 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.

authenticator

A record containing information that can be shown to have been recently generated using 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 protocol

A formal procedure for establishing the authorization of principals with respect to protected objects. Authorization protocols supported by DCE Security include DCE Authorization and Name-Based Authorization.

authorization service

See **authorization protocol.**

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. Backup 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 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 cancelled 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 control program

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.

cell

1. The basic unit of operation in the 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 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**.

cellular mount point

DFS: A mount point that resides in a different cell from the fileset it references. It directs the Cache Manager to the cell in which the fileset is located. See also **mount point**.

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.

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 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-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: The client 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 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

The 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.

collapse

To remove the contents of a directory from the display (close it) using 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: A group of related commands. The DFS command suites are **bak, bos, cm,** 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.

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

A 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

Either a copy of an entry stored in other DSAs through bilateral agreement, 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 automatically restarts 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 service 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.

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**.

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**.

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**.

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, DCE.**

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. XDS: 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.

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 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 File Service, DCE (DCE 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.

Distributed Time Service (DTS), DCE

The Distributed Time Service synchronizes the clocks in networked systems.

DIT

See **Directory Information Tree**.

DN

See **Distinguished Name**.

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 **Distributed Time Service, DCE**.

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 value used to encrypt data so that only possessors of the encryption key can decipher it.

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 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.

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**.

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 Time Service responds by abruptly setting the system clock to the computed time, rather than gradually adjusting the clock.

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 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.

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; when using other file systems, filesets are equivalent in size to a partition.

Fileset Database machine

DFS: A server machine in a cell that houses the Fileset Location Database (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: A data structure that implements the fileset concept. It resides on the disk aggregate with all of the files in its fileset and records physical memory addresses for the files. It also records the fileset's size, quota, fileset ID number, and other information.

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 Fileset Location Database is maintained by the Fileset Location Server (FL Server).

Fileset Location Server (FL Server)

DFS: A server process that runs on Fileset Database machines and maintains the Fileset Location Database (FLDB), which tracks the location of all available 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 the 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.

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**.

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

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, modelled 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

The 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 Cell Directory Service (CDS) to access names in foreign cells. The GDA provides a connection to foreign cells through either the Global Directory Service (GDS) or the Domain Name System (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**.

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 Attribute Configuration File (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 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 distinguished name is formed by appending its RDN to the distinguished name 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 distinguished name, followed by the RDN of the other, forms the distinguished name of the other entry.

immediate superobject

One object that contains another among its attribute values.

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 MB1 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.

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 Network Computing Architecture (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 Universal Unique Identifier (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 universal unique identifier generated for an RPC interface definition using the UUID generator, **uuidgen.** See also **interface definition, RPC interface, Universal Unique Identifier (UUID).**

intermediate data type

Any of the basic data types in terms of which the other, substantive data types of the interface are defined.

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.

IP

Internet Protocol. A family of network protocols defined by the U.S. Department of Defense (DoD).

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 file

DFS: A file that stores the unique server encryption keys used by the DFS server processes on a DFS server machine. Each DFS server machine has its own unique key file. See also **encryption key**.

Key Management Facility

A DCE Security facility that enables noninteractive principals to manage their secret keys.

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.

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

See **Local File System**.

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 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. The DCE LFS supports multiple filesets within a single aggregate, fileset replication, fast system restarts, and DCE Access Control Lists.

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 **/.:** 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 Local Area Network (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 entry point vector 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 access control lists, 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.

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

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.

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. If 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**.

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.

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 Remote Procedure Call 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 Network Computing Architecture. 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 Internet Protocol (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 translator

DFS: A utility that allows users working on NFS client machines to access the DFS file space.

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.

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 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.

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.

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**.

partially bound binding handle

RPC: A server binding handle that contains an incomplete server address lacking an endpoint. See also **fully bound binding handle**.

password

A string presented by a principal, which the Authentication Service uses to authenticate that principal. In addition to user-specific passwords, a user may also be required to enter a group-specific password for the purposes of authenticating membership in a group.

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.

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.

plaintext

The input to an encryption function or the output of a decryption function. Decryption transforms ciphertext into plaintext.

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

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.

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 name

The string name of an object to which any aliases for that object refer. The 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 the 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 the 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

A long-lived encryption key known to only one principal. In the DCE, the Authentication Service is the only principal that has a private key.

private object

1. XDS: An OM object created in a workspace 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 the 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 access control list.

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 three services provided by DCE Security; the Privilege Service certifies a principal's privileges. The other services are the Registry Service and the Authentication Service.

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 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**.

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 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. 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.

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 the 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 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 executor

DFS: The DFS client machine that executes DFS-specific system calls on behalf of an NFS client machine.

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 containing 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**.

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, in conjunction with the Cache Manager, tracks the currency of replicas and updates the version of data being used at the replication sites. 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 operation need not involve recovery from other media such as tapes. The DFS Backup System allows several different types of restores, including full restores and date-specific restores. 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 **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 Daemon (RPCD)

RPC: The process that provides the endpoint map service for a system. The RPC Daemon is started by the rpcd command. See also **endpoint map**, **endpoint map service**.

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**.

RPCD

See **RPC Daemon**.

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; it 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.

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 copies of the 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 the DCE, each secret key is known to the Authentication Service and one other principal.

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.

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 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 using the same session OM object.

session key

Used in Kerberos specifications; acronym for "conversation key." See also **conversation key**.

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 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 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.

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

Stub Programming Interface. A private RPC runtime interface whose routines are unavailable to application code.

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.**

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 the database and distributes them to the other, 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 to execute a command.

Tape Coordinator machine

DFS: A client machine from which backup and restore operations are initiated in 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.

TCID

See **Tape Coordinator ID**.

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

1. An application-transparent mechanism that transmits the identity of one principal to another.

2. 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.**

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**.

translator machine

DFS: A DFS client machine that is also an NFS server. The machine provides DFS with access to NFS client machines. See also **client machine**.

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 Fileset Location Database 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 Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP).

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 Fileset Location Server and the DFS Backup Server use to keep individual copies of their databases synchronized. See also **secondary site**, **synchronization site**.

UFS

See **UNIX File System**.

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**.

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. See also **illegal**.

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 type. 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 authentication cache module

DFS: The part of the DFS Cache Manager that maintains per-user Kerberos tickets and credential information.

user data

DFS: The data in a fileset or aggregate, such as applications and data files, created and referenced by users of the system.

User Datagram Protocol (UDP)

A protocol of the Internet Protocol (IP) family.

UTC

See **Coordinated Universal Time**.

UTS

See **Update Timestamp**.

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**.

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**.

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.

XDS

X/Open Directory Service.

XOM

X/Open Object Management.

Index

Symbols

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

X

OPEN SOFTWARE FOUNDATION™

INFORMATION REQUEST FORM

Please send to me the following:

( ) OSF™ Membership Information

( ) OSF™DCE License Materials

( ) OSF™DCE Training Information

Contact Name _____

Company Name _____

Street Address _____

Mail Stop _____

City _____ State _____ Zip _____

Phone _____ FAX _____

Electronic Mail _____

MAIL TO:

Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142

Attn: OSF™DCE

For more information about OSF™DCE call OSF Direct Channels at 617 621 7300.

OSF™DCE

Introduction to OSF™DCE

TITLES IN THE OSF™DCE SERIES:

Introduction to OSF™DCE

OSF™DCE User's Guide and Reference

OSF™DCE Administration Reference

OSF™DCE Application Development Guide

OSF™DCE Application Development Reference

Printed in the U.S.A.

ISBN 0-13-490624-1

9 780134 906249 90000

Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142

P T R Prentice-Hall, Inc.