XREFPAGE=NO.

IBM Distributed Computing Environment 2.1 for AIX: Administration Guide

Document Number XXXX-XXXX-XX

July 9, 1996

Note: Before using this information and the product it supports, be sure to read the general information under “Notices” on page xv.

Second Edition (June 1996)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

The information contained within this document is subject to change without notice.

OSF MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

OSF shall not be liable for errors contained herein, or for any direct or indirect, incidental, special or consequential damages in connection with the furnishing, performance, or use of this material.

Copyright © 1994 Open Software Foundation, Inc.

This documentation and the software to which it relates are derived in part from materials supplied by the following:

- © Copyright 1990, 1991 Digital Equipment Corporation
- © Copyright 1990, 1991 Hewlett-Packard Company
- © Copyright 1989, 1990, 1991, 1992, 1993, 1994 Transarc Corporation
- © Copyright 1990, 1991, 1993 Siemens Nixdorf Informationssysteme AG
- © Copyright 1990, 1995 International Business Machines Corporation
- © Copyright 1988, 1989 Massachusetts Institute of Technology
- © Copyright 1988, 1989 The Regents of the University of California.

All Rights Reserved

Printed in the U.S.A.

THIS DOCUMENT AND THE SOFTWARE DESCRIBED HEREIN ARE FURNISHED UNDER A LICENSE, AND MAY BE USED AND COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. TITLE TO AND OWNERSHIP OF THE DOCUMENT AND SOFTWARE REMAIN WITH OSF OR ITS LICENSORS.

- Open Software Foundation, OSF, the OSF logo, OSF/1, OSF/Motif, and Motif are trademarks of the Open Software Foundation, Inc.
- UNIX is a registered trademark of UNIX System Laboratories, Inc. in the U.S. and other countries.
- DEC, DIGITAL, and ULTRIX are registered trademarks of Digital Equipment Corporation.
- DECstation 3100 is a trademark of Digital Equipment Corporation.
- HP, Hewlett-Packard, and LaserJet are trademarks of Hewlett-Packard Company.
- Network Computing System and PasswdEtc are registered trademarks of Hewlett-Packard Company.
- AFS and Transarc are registered trademarks of the Transarc Corporation.
- Episode is a trademark of the Transarc Corporation.
- AIX and RISC System/6000 are trademarks of International Business Machines Corporation.
- IBM is a registered trademark of International Business Machines Corporation.
- DIR-X is a trademark of Siemens Nixdorf Informationssysteme AG.
- MX300i is a trademark of Siemens Nixdorf Informationssysteme AG.
- NFS, Network File System, SunOS and Sun Microsystems are trademarks of Sun Microsystems, Inc.
- X/OPEN is a trademark of the X/Open Company Limited in the U.K. and other countries.
- PostScript is a trademark of Adobe Systems Incorporated.

FOR U.S. GOVERNMENT CUSTOMERS REGARDING THIS DOCUMENTATION AND THE ASSOCIATED SOFTWARE

These notices shall be marked on any reproduction of this data, in whole or in part.

NOTICE: Notwithstanding any other lease or license that may pertain to, or accompany the delivery of, this computer software, the rights of the Government regarding its use, reproduction and disclosure are as set forth in Section 52.227-19 of the FARS Computer Software-Restricted Rights clause.

RESTRICTED RIGHTS NOTICE: Use, duplication, or disclosure by the Government is subject to the restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013.

RESTRICTED RIGHTS LEGEND: Use, duplication or disclosure by the Government is subject to restrictions as set forth in paragraph (b) (3)(B) of the rights in Technical Data and Computer Software clause in DAR 7-104.9(a). This computer software is submitted with "restricted rights." Use, duplication or disclosure is subject to the restrictions as set forth in NASA FAR SUP 18-52.227-79 (April 1985) "Commercial Computer Software-Restricted Rights (April 1985)." If the contract contains the Clause at 18-52.227-74 "Rights in Data General" then the "Alternate III" clause applies.

US Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

Unpublished — All rights reserved under the Copyright Laws of the United States.

This notice shall be marked on any reproduction of this data, in whole or in part.

Contents

Notices . . . xv
Trademarks and Service Marks . . . xv

About This Book . . . xvii
Who Should Use This Book . . . xvii
How This Book is Organized . . . xvii
Related Publications . . . xviii
Conventions Used in This Book . . . xviii

Chapter 1. DCE Introduction and Overview . . . 1-1
DCE for Administrators Overview . . . 1-1
DCE Control Program Overview . . . 1-22
DCE Administrative Tasks Overview . . . 1-26
DCE Application and Host Administration Overview . . . 1-27
Cell Directory Service (CDS) Overview . . . 1-28
Distributed Time Service (DTS) Overview . . . 1-34
DCE Security Overview . . . 1-44
Reference . . . 1-59

Chapter 2. The DCE Control Program . . . 2-1
Command Substitution Usage . . . 2-1
Controlling Your Script . . . 2-1
Convenience Variable Usage . . . 2-5
Creating Commands Dynamically with the **eval** Command . . . 2-10
Creating New Commands With the **proc** Command . . . 2-10
Customizing **dcecp** Sessions . . . 2-13
DCE Administration Objects . . . 2-15
DCE Control Program Usage . . . 2-15
dcecp Commands Usage . . . 2-19
dcecp Help Facilities Usage . . . 2-21
Documenting Your Scripts with Comments . . . 2-23
Editing Command Lines . . . 2-24
Error and Exception Handling . . . 2-26
Formal Task Objects . . . 2-29
Grouping Elements and Controlling Interpretation . . . 2-37
Informal Administration Scripts . . . 2-39
Interactive Command or Script Usage . . . 2-40
Looking Beyond the Tools . . . 2-41
Manipulating Strings . . . 2-41
Measuring and Counting with Expressions . . . 2-44
Operating on Lists . . . 2-45
Reading Other Files as **dcecp** Scripts with **source** Command . . . 2-46
Spawning Subprocesses . . . 2-47
Variable Substitution . . . 2-48
Working with Files . . . 2-49

Chapter 3. DCE Administration Tasks . . . 3-1
Backing Up the Security Service Registry and CDS . . . 3-1
Changing the IP Address . . . 3-2
Configuring a DCE Host in a Cell . . . 3-3

Creating a New User . . . 3-3
Creating, Maintaining, and Deleting Keytab Files . . . 3-5
Deleting a User . . . 3-9
Displaying All Configured DCE Servers and DCE Hosts . . . 3-10
Displaying All Servers Configured for a DCE Host . . . 3-11
Displaying User Information . . . 3-11
Listing the DCE Hosts in a Cell . . . 3-12
Modifying or Extending the Cell Object . . . 3-13
Modifying or Extending the Host Object . . . 3-13
Modifying or Extending the User Object . . . 3-13
Removing a DCE Host from a Cell . . . 3-14
Setting Up a Password Synchronization Server . . . 3-15
Setting Up a Password Strength Server . . . 3-17
Starting Configured DCE Processes on a Host . . . 3-20
Stopping DCE Processes Running on a Host . . . 3-20
Testing Cell Operation . . . 3-21
Testing Whether a DCE Host is Running . . . 3-22
Using Task Objects to Simplify DCE Administration . . . 3-22

Chapter 4. DCE Host and Application Administration . . . 4-1
Client Administration . . . 4-1
Controlling Server Operation . . . 4-3
DCE Host Services . . . 4-13
Easing Application Development and Administration . . . 4-14
Managing Client/Server Binding Information . . . 4-17
Managing Host Data . . . 4-18
Managing Server Entries, Groups, and Profiles in CDS . . . 4-21
Monitoring Your Program's Performance . . . 4-32
Routing Serviceability Messages . . . 4-33

Chapter 5. Cell Directory Service (CDS) . . . 5-1
ACL Entry Types Used for Principals . . . 5-1
ACL Types Supported by CDS . . . 5-2
An In-Depth Analysis of DCE Names . . . 5-2
Backing Up Namespace Information . . . 5-5
CDS Control Program Commands and Required Permissions . . . 5-6
CDS Operations . . . 5-8
CDS Servers Access to the Namespace . . . 5-10
cdscp Program Usage . . . 5-10
Cell Directory Service User Interface . . . 5-11
Cells Determine Naming Environments . . . 5-12
Controlling Access to CDS Clerk and Server Management Operations . . . 5-14
Creating Directories . . . 5-15
Creating a Read-Only Replica . . . 5-17
DCE Permissions Supported by CDS . . . 5-20
Deleting a Clearinghouse . . . 5-21
Deleting Directories . . . 5-22
Deleting Read-Only Replicas . . . 5-24
Directory Services Usage by DCE Components . . . 5-25
Directory Services and the Cell Environment . . . 5-25
Directory Service Usage . . . 5-27
Disabling Clerks and Servers . . . 5-28
Displaying the Contents of a Clearinghouse . . . 5-28
Displaying the Namespace with the CDS Browser . . . 5-28

Editing ACLs on CDS Names 5-30
Enabling Other Cells to Find Your Cell 5-31
Error Handling 5-33
Forcing the Clearinghouse to Checkpoint to Disk 5-34
Global Directory Agent Operation 5-34
Keeping Data Consistent Through Timestamps 5-37
Listing the Contents of Directories 5-37
Locating Names 5-40
Managing CDS with **dcecp** Commands 5-42
Managing Soft Links 5-43
Managing the Global Directory Agent 5-45
Merging CDS Directories 5-46
Merging CDS Directories into a Foreign Cell 5-49
Modifying a Directory's Convergence 5-50
Modifying a Directory's Replica Set 5-50
Monitoring Clerk Communications with Specific Clearinghouses 5-53
Monitoring Clerk, Server, and Clearinghouse Counters 5-54
Overview of DCE Authorization for CDS 5-54
Overview of the Merge and Append Procedures 5-55
Preserving a Clearinghouse Across a Server System Upgrade 5-56
Propagating Permissions to CDS Directories and Their Contents 5-56
Relocating a Clearinghouse 5-57
Restarting Clerk and Server Processes 5-59
Restoring Merged CDS Directories 5-60
Security in the Cell Directory Environment 5-61
Setting Up Access Control in a New Namespace 5-62
Skulk Operation 5-63
Skulking a Directory 5-64
Translating from Names to Resources 5-65
Update Propagation 5-69

Chapter 6. DCE Distributed Time Service 6-1
Changing the System Time 6-1
Configuring DTS for a LAN 6-3
Configuring DTS for an Extended LAN 6-3
Controlling Access to DTS 6-4
DTS Advantages 6-4
DTS Configuration for WANs and WAN Links 6-6
DTS Operations 6-9
DTS Planning for External Time Providers 6-11
DTS Timestamp Format 6-12
General DTS Planning Guidelines 6-12
Getting the Time from NTP Time Sources 6-13
Giving the Time to NTP Nodes 6-15
Managing DTS with **dcecp** Commands 6-17
Management Tasks Specific to Servers 6-19
Modifying Clerk and Server Attributes 6-21
Preventing Loops 6-28
Reconfiguring DTS on Nodes 6-30
Temporarily Reconfiguring DTS 6-31

Chapter 7. DCE Security Service 7-1
ACL Entries and Masks 7-1
ACL Management Tasks 7-9

Adding Accounts . . . 7-9
Adding and Maintaining Groups and Organizations . . . 7-14
Adding and Maintaining Principals . . . 7-16
Administration and Programming in DCE Audit . . . 7-19
Adopting Registry Orphans . . . 7-22
AIX/DCE Security Integration . . . 7-24
Authentication Policy . . . 7-33
Automatic Changes to the Locksmith Account . . . 7-35
Backing Up and Restoring the Registry Database . . . 7-36
Changing the Master Replica Site . . . 7-37
Changing the Registry Master Key . . . 7-38
Container ACLs . . . 7-39
Controlling Access to the Audit Daemon . . . 7-42
Controlling and Displaying Audit Trails . . . 7-44
Copying ACLs . . . 7-45
Creating and Maintaining Aliases for Principals and Groups . . . 7-46
Creating and Maintaining Attribute Instances . . . 7-47
Creating and Maintaining Attribute Types . . . 7-49
Creating and Maintaining Filters . . . 7-54
Creating Slave Replicas . . . 7-56
Creating the Primary Registry Database . . . 7-56
Creating Trust Relationships . . . 7-60
Defining Attribute Trigger Servers . . . 7-62
Defining Event Classes . . . 7-65
Displaying Account Information . . . 7-66
Displaying ACL Information . . . 7-68
Displaying Group and Organization Information . . . 7-69
Displaying **keytab** Information . . . 7-70
Displaying Principal Information . . . 7-71
Displaying **xattrschema** Information . . . 7-73
Enabling and Disabling the Audit Logging Service . . . 7-74
Ensuring Consistent Local Files . . . 7-74
Establishing Trust Relationships . . . 7-75
Extended Security Attributes for Principals . . . 7-79
Forcibly Deleting a Slave Replica . . . 7-83
Handling Network Address Changes . . . 7-84
How Identities Represented by Accounts are Authenticated . . . 7-85
How the Registry Database is Stored . . . 7-88
Initial Registry ACLs . . . 7-89
Machine Accounts . . . 7-90
Maintaining Membership Lists . . . 7-90
Maintaining the Local Registry . . . 7-92
Modifying and Querying Audit Daemon Attributes . . . 7-94
Modifying Cross-Cell Authentication Accounts . . . 7-94
Object Creation Quotas . . . 7-95
Overriding Entries in the Local /etc/passwd and /etc/group Files . . . 7-95
passwd_import Operation . . . 7-100
passwd_import Usage . . . 7-102
Physical Security of the Database . . . 7-103
Planning Sites for Security Service Components . . . 7-103
Policies . . . 7-104
Populating the New Registry Database . . . 7-106
Principal, Group, and Organization Names . . . 7-106
Properties . . . 7-108

Recovering Slave Replicas . . . 7-110
Recovering the Master Replica . . . 7-111
Recovering the Registry . . . 7-112
Registry ACL Manager . . . 7-112
Registry Editor Usage . . . 7-113
Registry Permissions . . . 7-114
Removing a Server Machine from the Network . . . 7-125
Replicated Databases . . . 7-126
Reserved Principals and Accounts . . . 7-129
Restarting a Security Server in Locksmith Mode . . . 7-130
Restarting Security Servers . . . 7-130
Restarting the Primary Server in Locksmith Mode . . . 7-131
Restoring a Duplicate Master . . . 7-131
Sample **passwd_import** Session . . . 7-131
Security Object Names . . . 7-135
sec_admin Program Usage . . . 7-136
Server Accounts . . . 7-137
Setting DCE Audit Environment Variables . . . 7-138
Setting the **_s(sec)** Variable . . . 7-138
Setting Up the Registry . . . 7-139
Starting a Security Server in Locksmith Mode . . . 7-140
Starting the Audit Daemon . . . 7-140
Starting the Master Replica . . . 7-141
The /**etc**/**passwd** and /**etc**/**group** Files and the Registry . . . 7-141
The Local Registry . . . 7-141
The **passwd_import** Command Syntax . . . 7-141
The Registry Database . . . 7-142
The **xattrschema** Object . . . 7-143
Trust Relationships . . . 7-143
Turning a Master Replica into a Slave . . . 7-145
Universal Unique Identifiers and UNIX IDs . . . 7-145
User Accounts . . . 7-146
Validating the Authenticity of the DCE Security Service . . . 7-146
Verifying that Replicas are Running . . . 7-146

Appendix A. Reference . . . A-1
Additional Rules . . . A-1
Criteria for Selecting a Time Source . . . A-2
DTS Extended BNF . . . A-2
File Locations . . . A-3
Maximum Name Sizes . . . A-5
Metacharacters . . . A-5
Modifying CDS Entity Attributes . . . A-6
Modifying the **cds_attributes** and **cds_globalnames** Files . . . A-8
Origin of Object Identifiers . . . A-8
Sources of Coordinated Universal Time . . . A-8
The **cds_attributes** File . . . A-9
The **cds_globalnames** File . . . A-11
The CDS Space . . . A-12
The Security Space . . . A-27
Valid Characters and Naming Rules for CDS . . . A-48
World Time Zone Map . . . A-50

Index . . . X-1

Figures

1-1. Interaction of Clients and Servers . . . 1-2
1-2. Top Level of the Cell Namespace . . . 1-10
1-3. Components of a CDS Server Node . . . 1-33
1-4. Time and Inaccuracy . . . 1-37
1-5. Computed Time . . . 1-38
1-6. Adjustment of the Clock . . . 1-39
1-7. ISO-Compliant Time Format . . . 1-40
1-8. ISO-Compliant Time Format Variation . . . 1-41
1-9. Relative Time Format . . . 1-41
1-10. ACL Managers in Servers . . . 1-46
1-11. Event Class Number Formats . . . 1-53
1-12. Override Relations Between Filter Types . . . 1-56
1-13. Machines, Servers, and the Database . . . 1-56
4-1. Server Binding Information . . . 4-18
4-2. Possible Information in a Server Entry . . . 4-23
4-3. Possible Mappings of a Group . . . 4-25
4-4. Possible Mappings of a Profile . . . 4-28
5-1. Sample CDS Namespace Hierarchy . . . 5-3
5-2. Sample Portion of the BIND Namespace . . . 5-4
5-3. CDS Clerks and Servers on a LAN . . . 5-9
5-4. A Sample CDS Lookup . . . 5-9
5-5. Cell and Global Naming Environments . . . 5-26
5-6. Interaction of CDS, GDA and Global Directory Services . . . 5-27
5-7. CDS Browser Icons and Their Meaning . . . 5-29
5-8. How the CDS Clerk Finds a GDA . . . 5-35
5-9. How the GDA Helps CDS Find a Name . . . 5-36
5-10. How the Clerk Finds a Name . . . 5-41
5-11. Example Namespace Before and After the Merge Operation . . . 5-48
5-12. Example Namespace Before and After the Append Operation . . . 5-49
5-13. Example Replica Set . . . 5-52
5-14. Example Replica Set After Master Redesignation . . . 5-52
5-15. Example Replica Set After Replica Exclusion . . . 5-53
5-16. An Example of Namespace Hierarchy . . . 5-56
5-17. Logical and Physical Views of a Namespace . . . 5-66
5-18. Clearinghouse Object Entries and Clearinghouses . . . 5-67
5-19. A Soft Link and Its Resolution . . . 5-68
5-20. Child Pointers and Directories . . . 5-69
6-1. DTS Configuration — LAN . . . 6-3
6-2. DTS Configuration — LAN with WAN Links . . . 6-7
6-3. DTS Configuration — WAN Networks . . . 6-8
6-4. DTS Timestamp Syntax . . . 6-12
6-5. Local Time Source . . . 6-14
6-6. Getting the Time from a Remote NTP Time Source (Scenario 1) . . . 6-15
6-7. Getting the Time from a Remote NTP Time Source (Scenario 2) . . . 6-15
6-8. Giving the Time to NTP . . . 6-17
6-9. Local Fault . . . 6-25
6-10. Configuration Before Stratum 2 Node Fails . . . 6-29
6-11. Configuration After Stratum 2 Node Fails . . . 6-29
7-1. Order of Checking ACLs and Applying Masks . . . 7-7
7-2. Sample ACL Entries . . . 7-8

7-3. Initial ACLs for Objects Created in Containers 7-40
7-4. Initial ACLs for Containers Created in Containers 7-40
7-5. Disk Memory and Virtual Memory Copies of the Registry Database 7-89
7-6. The Master Replica Update Process 7-127
7-7. The Slave Replica Update Process 7-128
7-8. The Registry Database Structure 7-143
7-9. Transitive Trust Relationships 7-144
A-1. The Top-Level CDS Directory A-12
A-2. The CDS hosts Directory A-13
A-3. The Host Daemon Directory A-13
A-4. The CDS subsys Directory A-13
A-5. The Top-Level Security Directory A-28
A-6. The sec/group Directory A-28
A-7. The sec/principal Directory A-29
A-8. Valid Characters in CDS, GDS, and DNS Names A-49
A-9. World Time Zone Map A-51

Tables

4-1. Serviceability Message Severity Levels 4-34
5-1. Permissions Required to Create Target Objects 5-34
7-1. Stricter Standard Policies 7-34
7-2. Locksmith Account Changes Made by the Security Server 7-35
7-3. Registry Policy Changes Made by the Security Server 7-36
7-4. Initial Persons, Groups, and Organizations 7-58
7-5. DCE for AIX 2.1/Pre-DCE for AIX 2.1 Authentication Interoperation 7-82
7-6. ACL Managers, Valid Permissions, and ACL Entry Types 7-113
A-1. Summary of CDS, GDS, and DNS Characteristics A-1
A-2. Time Provider Selection Criteria A-2
A-3. Maximum Sizes of Directory Service Names A-5
A-4. Metacharacters and Their Meaning A-6

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make them available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program, or service that does not infringe on any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue Thornwood, NY 10594 USA.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*) and used in this publication, are trademarks or service marks of IBM Corporation in the United States or other countries:

AIX OS/2 IBM

The following terms, denoted by a double asterisk (**) and used in this publication, are trademarks of other companies as follows:

- OSF/Motif is a trademark of the Open Software Foundation, Inc.
- UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.
- Network Computing System is a registered trademark of Hewlett-Packard Company.

About This Book

The objective of this book is to assist system and network administrators in managing the Distributed Computing Environment (DCE) services provided by IBM* DCE for AIX*.

DCE for AIX offers the following services:

- Remote Procedure Call
- Directory Service
- Distributed Time Service
- Security Service

Most administrative functions may be performed on any node in a distributed environment. This book assumes that these tasks will be done on IBM DCE for AIX* systems only.

Planning, installing, and configuring the DCE services are often part of the administrator's responsibilities; however, these topics are not covered by this guide. For information on these topics, see *IBM Distributed Computing Environment 2.1 for AIX: Up and Running!*.

Who Should Use This Book

This book is intended for administrators. Although the basic DCE principles relating to administration are discussed at the beginning of each part, it is recommended that you have basic knowledge of DCE prior to using this guide.

Some understanding of the UNIX** operating system and TCP/IP communications will also help you use this guide effectively.

How This Book is Organized

This guide is comprised of two main types of information — introductory and tasks.

- Chapter 1, “DCE Introduction and Overview” on page 1-1 provides introductory information on the DCE tasks normally performed by an administrator. Since this section also groups together related tasks in the order in which they may normally occur, you can access these referenced tasks in a logical, tutorial-type manner. You may find this helpful in performing a set of tasks together from beginning to end.
- Specific tasks are grouped by programs and services. These tasks are presented alphabetically within the following sections:
  - Chapter 2, “The DCE Control Program” on page 2-1 describes control program tasks, utilities, and processes.
  - Chapter 3, “DCE Administration Tasks” on page 3-1 describes administrative tasks used to manage cells, hosts, and users.
  - Chapter 4, “DCE Host and Application Administration” on page 4-1 describes host and application administration within DCE.
  - Chapter 5, “Cell Directory Service (CDS)” on page 5-1 describes cell creation and maintenance tasks such as clerk and server access, replica management, and security considerations.
  - Chapter 6, “DCE Distributed Time Service” on page 6-1 describes clock synchronization processes and related tasks.

– Chapter 7, “DCE Security Service” on page 7-1 describes security-related tasks in the Distributed Computing Environment.

Related Publications

In order to obtain a good foundation of the components and concepts behind DCE prior to performing administrative tasks using this guide, you may find it helpful to read or refer to the following:

- *Directory and Security Server for AIX: Up and Running!*
- *IBM Distributed Computing Environment 2.1 for AIX:Introduction to Distributed Computing Environment*
- *IBM Distributed Computing Environment 2.1 for AIX:Administration Command Reference*
- *IBM Distributed Computing Environment 2.1 for AIX:Application Development Guide — Introduction and Style Guide*
- *IBM Distributed Computing Environment 2.1 for AIX:Application Development Guide — Core Components*
- *IBM Distributed Computing Environment 2.1 for AIX:Application Development Guide — Directory Services*
- *IBM Distributed Computing Environment 2.1 for AIX:Application Development Reference*
- *IBM Distributed Computing Environment 2.1 for AIX:Network File System to Distributed File System Authenticating Gateway Guide and Reference*
- *IBM Distributed Computing Environment 2.1 for AIX:DFS Administration Guide and Reference*

Conventions Used in This Book

This book uses the following typographic conventions:

Bold	**Bold** words or characters represent system elements that you must enter into the system literally, such as commands and file, path, and directory names.
Italics	*Italicized* words or characters represent values for variables that you supply. Italics is also used to highlight a word or phrase that is being described.
`Example Font`	Examples and information displayed by the system are printed using an example font that is a `constant width typeface`.
[ ]	Optional items found in format and syntax descriptions are enclosed in brackets.
{ }	In format and syntax descriptions, a list from which you must choose an item is enclosed in braces.
|	A vertical bar separates items in a list of choices.
< >	Angle brackets enclose the name of a key on a keyboard.
...	Horizontal ellipsis points indicate that you can repeat the preceding item one or more times. Vertical ellipsis points indicate that you can repeat the preceding item one or more times.
/opt/dcelocal	Represents the environment variable that specifies the drive and path to the top of a directory containing the DCE product code on your local machine. For more information, see *DSS for AIX Up and Running!*.

Chapter 1. DCE Introduction and Overview

This section provides an introduction to the DCE tasks normally performed by an administrator. It also provides an overview of the various components used to administer DCE.

DCE for Administrators Overview

This section provides an overview of DCE from the perspective of the system or network administrator.

As the *Introduction to Distributed Computing Environment* explains, DCE is a set of services that together make up a high-level coherent environment for developing and running distributed applications. These services include a set of tools that support DCE management tasks. DCE applies techniques that you may have learned from working with applications for single machines or other distributed systems. These techniques enable system administrators to manage DCE without having to know about system internals. You can start with a configuration that is appropriate for your initial needs and grow to larger configurations without sacrificing reliability or flexibility. DCE supports large networks with many users, as well as smaller networks.

Introduction

The following concepts, which are described in the remaining topics of this section, are central to DCE system administration:

- Clients and servers to make and respond to requests for a service
- Remote Procedure Calls (RPCs) for client-to-server communications
- Cells, which are groups of users, servers, and machines that share security, administrative, and naming boundaries
- A single namespace that lets client applications identify, locate, and manage objects, including users, machines, servers, groups of servers, and directories
- A single filespace that allows data sharing among users and machines with proper authorization
- Principals, which are entities—including users, servers, and computers—that are capable of communicating securely with other entities
- Access Control Lists (ACLs) to control access to objects
- Caching, which is the technique of copying a file from a File Server machine (its central storage place) to a client machine's local disk or memory where users can access the copy locally
- Replication, which is the process by which copies of information are created and kept consistent

Clients and Servers

DCE is based on the client/server model. A server is a machine or process that provides a specialized service to other machines or processes. A client is a machine or process that uses the server's specialized service during the course of its own work. Distributed applications consist of a client side that initiates a request for service, and a server side that receives and executes that request, and returns any results to the client. For example, a client can request that a file be printed, and the server where the printer resides carries out that request.

More than one server process can reside on a single machine. Also, one machine can be both a client and a server. For example, a machine can be a client for one DCE component and a server for another.

Figure 1-1 on page 1-2 shows a machine that is a name server for a client that issues a name request. The same machine is a client for a file server.

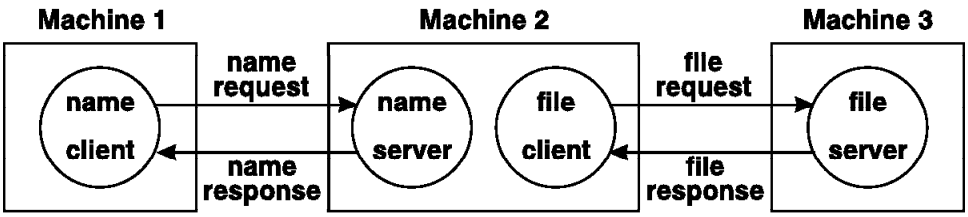


Figure 1-1. Interaction of Clients and Servers

Remote Procedure Call

A Remote Procedure Call (RPC) is a synchronous request and response between a local calling program and a remote procedure. An RPC begins with a request from a local calling program to use a remote procedure. It completes when the calling program receives all the results (or an error status or exception) from the procedure.

The Cell

The cell is the basic unit of administration in DCE. A cell usually consists of users, machines, and resources that share a common purpose and a greater level of trust with each other than with users, machines, and resources outside of the cell. Members of a cell are usually located in a common geographic area, but they can also be located in different buildings, different cities, or even different countries, provided they are adequately connected. A cell's size can range from only one machine to several thousand, depending on the size of the organization. All machines in an organization can be included in one cell, or you can choose to have numerous cells within one organization.

Cells designate security, administrative, and naming boundaries for users and resources. Each cell has a name. Cell names are established during the installation and configuration of DCE components.

Members of an organization who are working on the same project are likely to belong to the same cell. For example, in a large organization with several cells, the sales team could belong to one cell, the engineers working on Project X could belong to a second cell, and the engineers working on Project Y could belong to a third cell. On the other hand, a small organization may have only one cell for both the sales force and the engineers because they share the same level of security and the organization's small size does not warrant the additional administrative overhead that maintaining additional cells requires.

DCE services are managed within the context of a cell, as described by the following examples:

- Each DCE cell typically consists of at least one Cell Directory Service (CDS) server, three Distributed Time Service (DTS) servers, and one Security Service server, as well as the databases that the CDS and Security servers use.
- Pathnames of DCE objects managed by DCE services can be expressed relative to the cell where the objects reside.
- DTS has both local and global servers. Local servers operate within a Local Area Network (LAN). Global servers provide time services anywhere within the cell.

The Namespace

The namespace is the hierarchical set of names of DCE objects. The top levels of the hierarchy are managed by the Directory Service. Some DCE services (currently the Security Service and Distributed File Service (DFS)) manage their own portions of the namespace. Each DCE object in the namespace consists of a name with associated attributes (pieces of information) that describe its characteristics. These objects include resources such as machines or applications.

The namespace contains global namespaces and cell namespaces. A cell namespace includes objects that are registered within a cell. A logical picture of a cell namespace is a hierarchical tree with the cell root directory at the top and one or more levels of directories containing names beneath the cell root. The cell namespace is managed by the Cell Directory Service (CDS) component of the Directory Service. Conversely, the global namespace, as seen from a local DCE cell, contains objects that are registered outside the local cell, such as the names of other cells. The Global Directory Service (GDS) component of the Directory Service manages one part of the global namespace; a non-DCE service called the Domain Name System (DNS) manages another part of the global namespace.

Note: Although this release of DCE for AIX does not provide the Global Directory Service (GDS), this information is provided for your reference if GDS is available in your cell from another product (such as DCE 1.3 for AIX).

Administrative tools use the namespace to store information and to locate DCE services. DCE services advertise their locations to the namespace. The namespace provides a means of organizing DCE services into manageable groups.

The Filespace

The filespace is part of the cell namespace and consists of files and directories. These files and directories can be physically stored on many different machines, but are available to users on every machine, if those users have the proper authorization. You manage the filespace in units called *filesets*, which are hierarchical groupings of related files. Although files are distributed throughout the network, located on and managed by different servers, users see a single filespace. DCE provides administrative tools to assist you in backing up, moving, and replicating filesets.

See the *DSS for AIX Up and Running!* and *DCE for AIX DFS Administration Guide and Reference* for cell filespace planning guidelines.

Principals

A DCE principal is an identity that is authenticated by the Security Service. When you log into your system, you use your principal name. Principals can be organized into groups and into organizations that contain groups of principals. Information associated with a principal includes information that is traditionally kept in AIX group and password files, such as the username, group ID, members of a group, and a user's home directory. By default, a principal is known within the bounds of a cell. By creating a special account that indicates you trust another cell's authentication service, you can enable principals from other cells to participate securely within your cell. See “Establishing Trust Relationships” on page 7-75 for information on creating these special accounts.

Access Control Lists

An Access Control List (ACL) is an authorization mechanism that allows you to assign permissions that control access to DCE objects. The following DCE objects are protected by ACLs:

- Principals and groups of principals managed by the Security Service
- Files and file system directories managed by the DFS

- DTS servers
- CDS directories and entries
- CDS clients and servers, which have ACLs restricting the use of their management operations (for example, creating a clearinghouse)

An ACL consists of multiple ACL entries that define the following:

- Who can use an object
- What operations can be performed on the object

In the filespace, ACLs are an extension of the AIX system file protection model. While AIX file system permissions are limited to the protection of files and directories, DCE ACLs can also control access to other objects, such as individual database entries, objects registered in the cell namespace, and objects managed by applications.

Caching

Information acquired over the network (for example, through the use of DCE RPC) can be stored in a memory or disk cache on the local machine. This technique reduces network load and speeds up lookups of frequently needed data. For example, information about the namespace and the filespace is cached by DCE client machines.

Caching can be configured on a service-by-service basis. Different caching mechanisms are used for different components of DCE. Each component has configurable options to improve the performance of your installation.

Replication

Replication increases the availability of resources by having copies of the resource on several machines. For example, with replication you can make database updates on one machine and have them automatically made on other machines in the network. You can replicate data, move replicas, and control the frequency of updates. The Security Service, CDS, and DFS all provide replication facilities that are customized for their particular applications.

Global and Cell Considerations

The purpose of this guide is to assist you in planning for the installation, configuration, and maintenance of DCE. DCE provides System Management Interface Tool (SMIT) and configuration utilities to assist you. For detailed information about installing the DCE source tape and building DCE, refer to the *Release Notes*. *DSS for AIX Up and Running!* describes the configuration process, including installing executable files, setting up a DCE cell, and configuring servers and clients.

This section discusses how to establish a DCE cell name, describes how the cell namespace is organized, and provides guidelines for maintaining security and replicating parts of the cell namespace. See *DSS for AIX Up and Running!* and *DCE for AIX DFS Administration Guide and Reference* for a discussion of specific DFS considerations as you plan for including DFS in your cell.

You need to answer a number of questions when planning for a distributed system. Your answers to these questions determine the basic requirements of your user environment. Keep in mind the following global considerations as you plan for DCE:

- **How much do you think your environment will grow in the next few years? Do you anticipate rapid or relatively slow expansion of your network?**

If you think your environment will grow rapidly, consider setting up several cells representing smaller units of your organization. You can manage these smaller units as your network expands. As explained in the *Introduction to Distributed Computing Environment*, members of each cell share a common purpose, and the cell is a unit of administration and security. If you anticipate slow expansion of your network, you may be able to establish one or more cells based on the organization that exists now. Consider how many administrators you will need to maintain your DCE cell, based on anticipated future growth.

- **How much information does your environment have that needs to be distributed? How much do the users in your network share information?**

  If there is a large volume of information that needs to be shared within your network, consider the amount of disk space you require and the number of DFS File Server machines you need.

- **How much information updating do you require? Do the users in your network mainly look up information, or do they create and change information at their workstations?**

  If information changes frequently and users in your network depend on the accuracy of that information, you need to consider how much you rely on replication. It is better to go to a central source of information for data that changes frequently. If users look up information but do not need to change the information that is shared with other users, you can rely more on replicated data.

- **Is the most important data the most available? Have you made plans to replicate this data?**

  CDS, the Security Service, and DFS maintain master copies of their respective databases. Each CDS directory can be replicated separately. In addition to DFS databases, individual DFS filesets or groups of filesets can be replicated. The Security Service replicates the entire registry database. Because other components depend on the information managed by the Security Service and parts of the CDS namespace, that data needs to be available at all times. For example, the special character string **/.:** (the cell root) is stored in CDS and must always be available.

  Keep in mind that while replicating data helps availability, there is a cost in terms of performance and the amount of administration required.

- **If your network has a gateway, are the servers located on the same side of the gateway as the clients that rely on those servers?**

  CDS servers broadcast messages at regular intervals to advertise their existence to CDS clerks in the network. Clerks learn about servers by listening for these advertisements. Placing servers and the clients that rely on them on the same side of the gateway facilitates efficient updates of information and a quick response to client requests. Additional administration is required if you rely on servers that are not available through the advertisement protocol, which is effective only in a local area network.

  Consider how fast and how expensive links are if you are administering a cell that includes users in different geographic locations. You may want to keep more information locally to reduce your dependence on transmitting information across links.

- **Is communication limited to your own cell, or do you need to communicate with other cells?**

  DCE offers this method of connecting cells so that they can communicate:

  – The standard intercell connection, in which a cell is registered in a global directory service that DCE supports and communicates with other cells registered in that directory service. A cell can be registered in both the GDS and DNS directory services.

    Note: Although this release of DCE for AIX does not provide the Global Directory Service (GDS), this information is provided for your reference if GDS is available in your cell from another product (such as DCE 1.3 for AIX).

  In order for your cell to communicate with other cells, you must:

  – Establish a unique name for your cell and define it in the appropriate namespace (GDS or DNS),

- Have at least one GDA running in the cell, and
- Establish a Security Service trust relationship with the other cells with which you plan to communicate.

Establishing a Cell Name

You need to establish a name for your cell before you can begin the process of configuring it. A uniquely identified cell name is critical to the operation of the Security Service; this name is the basis for authentication in your cell. Whether or not your cell name needs to be globally unique depends on your plans for communication with other cells.

If you plan to create a private cell and do not ever intend for it to communicate with cells outside your organization, you are not required to obtain a globally unique cell name. However, if you plan to communicate with other cells, you need to obtain a globally unique cell name for your cell before you configure it.

If you plan to communicate with other cells through GDS or DNS, you need to obtain a globally unique name for your cell. The next sections describe how to establish GDS and DNS names for your cell. See "Valid Characters and Naming Rules for CDS" on page A-48 for a description of the valid characters supported in GDS, DNS, and CDS.

Note: GDS is not provided with DCE 2.1 for OS/2; however, this product can use GDS to register cells and to locate other cells if GDS is provided by another product (for example, DCE 1.3 for AIX).

Establishing a GDS Cell Name: Although this release of DCE for AIX does not provide the Global Directory Service (GDS), this information is provided for your reference if GDS is available in your cell from another product (such as DCE 1.3 for AIX).

If you plan to use GDS to communicate with other cells, you need to obtain a globally unique name for your cell from the GDS global naming authorities before you configure your cell, then define it in the GDS namespace. The name you obtain for your cell will be in GDS syntax.

As explained in the *Introduction to Distributed Computing Environment*, GDS-style names consist of a series of attribute/value pairs, separated by equals signs (=). Each attribute/value pair is called a Relative Distinguished Name (RDN). The Directory Information Tree (DIT) determines the hierarchy of a GDS name; that is, how the RDNs are ordered to create a global name. An example of a GDS-style global name, called a Distinguished Name (DN), is **/.../C=US/O=ABC/OU=DCE/CN=gunther**. Each RDN is separated by slashes (/).

DCE cell name values are generally stored in the Organization (**O=**) or Organization Unit (**OU=**) attributes of a GDS name. For example, the global name

```
/.../C=US/O=ABC/OU=Seattle
```

uses the **OU=** attribute to store the cell name **Seattle**. There is a fixed set of two-letter codes that must be used to indicate the Country (**C=**) attribute for your cell (for example, **C=US** or **C=JP**). The country in which you reside may also have standard Organizations (**O=**), where your organization is registered as a code. Your organization may also have conventions that apply to the way an Organization Unit (**OU=**) is represented. Check with your naming authority for exact conventions. Any valid X.500 name, including names provided by other standards supported by X.500, can be used as a cell name.

Note: A GDS style cell name cannot be a subset of another GDS style cell name. For example, if one cell had the name **/.../C=US/O=ABC**, a second cell could not use the name **/.../C=US/O=ABC/OU=Seattle**. This would cause a problem in intercell communications.

To obtain a unique GDS name for a cell, contact the administrator in charge of the portion of the DIT under which you want to name your cell. For example, in the United States, the American National Standards Institute (ANSI) delegates X.500 names that are subordinate to the RDN /**C=US**. Suppose you are an employee of ABC, a U.S. corporation interested in participating in a worldwide X.500 directory. If you wanted to configure a single cell whose name is /**C=US**/**O=ABC**, you would contact ANSI to reserve ABC as a unique organization name. Similarly, if you wanted to configure multiple cells in your organization and name them based on organization units, you would contact a naming authority within your company to establish a cell entry such as /**C=US**/**O=ABC**/**OU=Sales**.

Send X.500 name registration requests to

American National Standards Institute
11 West 42nd Street
New York, NY 10036

Telephone Number: (212) 642-4976

It is the responsibility of the person making the request to ANSI to be sure that your organization does not send more than one request for an organization name. After you receive your organization name, it is recommended that your organization set up a central administrative authority to manage names that are subordinate to the organization name.

After you have configured your cell, you need to define it in the GDS global namespace. GDS stores cell information in the **CDS-Cell** and **CDS-Replica** attributes. You must add these two attributes to an existing GDS entry for that entry to become a cell entry.

Establishing a DNS Cell Name: DCE also supports global directory operations through the use of DNS. If you plan to use DNS to communicate with other cells, you need to obtain a globally unique name for your cell from the DNS global naming authorities before you configure your cell, then define it in the DNS namespace. The name you obtain for your cell will be in DNS syntax. An example of a DNS-style cell name is:

```
/.../seattle.abc.com
```

If you plan to use DNS as your global directory service, your DCE cell name must follow the ARPA Internet Domain System conventions for site names. If you are already an Internet site, you can create one or more cells subordinate to your Internet domain name, depending on how your site is organized. The following conventions govern an Internet-style name:

- The name needs to have at least two levels; for example, **abc.com** or **sctech.edu**. The names in the first two levels are registered with the Network Information Center (NIC), which is the naming authority for DNS names.
- The name cannot contain more than 255 characters.
- The name can contain any number of fields in addition to the two required levels, which are conventionally separated by periods.
- The name needs to end in a suffix that indicates a kind of institution. This last field is the most significant one, in contrast to a GDS name, which begins with the most significant field. The standard suffixes are as follows:
  - **.com** for businesses and other commercial organizations
  - **.org** for noncommercial organizations
  - **.edu** for educational institutions
  - **.gov** for government institutions

- **.mil** for military institutions
- **.net** for network support organizations
- *.xx* for two-letter country codes (such as **.de** for Germany and **.fr** for France) that conform to the International Organization for Standardization (ISO)

For further information about naming rules, including valid characters, restrictions, metacharacters, and maximum name sizes for DNS names, see "Valid Characters and Naming Rules for CDS" on page A-48.

To obtain a unique DNS name, contact the administrator in charge of the subtree under which you want to name your cell. Send registration requests to the NIC at the following Internet address, telephone number, FAX number, or mailing address:

HOSTMASTER@INTERNIC.NET

Telephone Number: (703) 742-4777
between the hours of 7:00 a.m. and 7:00 p.m. Eastern Standard Time

FAX (703) 742-4811

Network Solutions, Inc., An SAIC Company
Attention: InterNIC Registration Services
505 Huntmar Park Drive
Herndon, VA 22070

After you have configured your cell, you need to define it in the DNS global namespace by creating a cell entry for it in DNS. To create a cell entry in DNS, an administrator must edit a data file that contains *resource records*.

You also need to establish cross-cell authentication with any other cells with which you want to communicate.

For further information, see "Modifying Cross-Cell Authentication Accounts" on page 7-94.

The Cell Namespace

An integral part of planning for a DCE cell is understanding the organization of your cell namespace. Consider the following as you plan the organization of a cell in your network:

- Are security requirements maintained?
- Does the organization of the cell facilitate network traffic where data sharing needs are the greatest?
- How will you manage the administrative accounts created for each DCE service during the configuration process?
- What are your DFS administrative domains (groups of DFS servers that are administered as a unit)? Can you group servers for more efficient administration?

Determining Cell Boundaries: In DCE, the boundaries of a cell are equivalent to the boundaries of the cell namespace. A small organization can consist of one cell. A large organization can have many cells. The primary factors in determining a cell's boundaries are the common purpose and trust shared by the cell's principals. Principals within a cell can belong to groups that share the same privileges. Members of a group share the same level of trust and are authorized to perform certain actions.

Because there is a set of administrative tasks associated with setting up and maintaining each cell, it is reasonable to keep the number of cells in your organization to a minimum. However, the level of trust shared by groups of principals is a more important consideration than administrative overhead.

Keeping Cells Stable: After you decide how many cells you need and where the boundaries of those cells will be, make an effort to keep your cell structure stable. Servers are not easily moved from one host to another, so be sure to plan your namespace structure carefully in order to minimize reconfiguration. If you do need to move a machine from one cell to another, you must:

- Use the configuration process to remove all configuration data from the machine.
- Reconfigure the host in the new cell.
- Delete any namespace or registry entries for the host in the old cell.

See *DSS for AIX Up and Running!* for configuration procedures.

Types of Cell Namespace Entries: The following sections describe the different types of entries that comprise the cell namespace. These entries are created when you follow the default configuration path described in *DSS for AIX Up and Running!*. This guide and the *DCE for AIX DFS Administration Guide and Reference* provide details about the names that the DCE components use.

The cell namespace can be divided into these major parts:

- The CDS part of the namespace
- The Security part of the namespace
- The DFS part of the namespace (the filespace)
- The **dced** (per host) part of the namespace

Each DCE service maintains its own namespace within the cell namespace. DFS maintains its own namespace to ensure consistency among many files. The Security Service maintains its own namespace to ensure that the DCE cell remains secure. Clients of these two services query CDS for binding information that enables them to find Security or DFS servers. The points where the binding information is stored serve as mount points in the CDS namespace for the namespaces that DFS and the Security Service manage. This transition point between two namespaces is called a *junction*. The /**.:/sec** directory is the junction from the CDS part to the Security part of the cell namespace, and the /**.:/fs** directory is the junction from the CDS part to the DFS part of the cell namespace.

The junction /**.:/hosts**/*hostname*/**config** is the junction from CDS to the **dced** (per host) part of the namespace.

Figure 1-2 on page 1-10 shows the top level of the cell namespace. In some cases the names in the cell namespace are fixed (or well known) and cannot be changed. In other cases you can choose a different name from the one listed. In Figure 1-2 on page 1-10, /**.:** and **cell-profile** are well-known names.

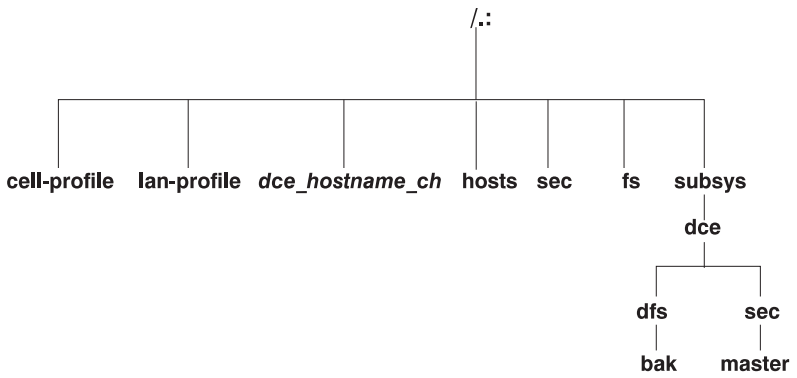


Figure 1-2. Top Level of the Cell Namespace

You can use the CDS browser (**cdsbrowser**) or the DCE control program (**dcecp**) to view the CDS part of the namespace, including the **sec** and **fs** junctions. You can use commands such as **ls** to see the contents of the DFS part of the cell namespace and the **dcecp** program to see the contents of the Security portion.

CDS Namespace Entries: The CDS namespace contains entries for servers, hosts, CDS clearinghouses (collections of directory replicas stored at a particular server), RPC profiles, RPC groups, and subsystems. The entries have a CDS type of **directory** or **object**, indicating the kind of CDS object to which the name refers. A third CDS type, **softlink**, is an entry that points to another entry. A CDS directory is a container in which objects are stored. CDS uses directories to organize groups of object entries.

In addition, the CDS namespace provides specialized services for other DCE components, such as location information contained in the fileset location database (FLDB), which is the database that maps filesets to the filesets to the file server machines on which they reside.

Profiles catalogued in the CDS namespace specify a search path through the Directory Service. The cell profile (/**.:/cell-profile**) stores the location of the servers that are available in the cell, regardless of physical location. In a geographically dispersed cell, servers can be located in different cities or even different countries. The LAN profile defines alternate servers that can be used in situations where geographic proximity is important. For example, /**.:/lan-profile** is the default LAN profile used by DTS. This profile contains entries for the DTS server local set. If a cell spans more than one LAN, another layer can be created below /**.:/lan-profile** to specify the location of the profile for each part of the cell. For example, in a cell that encompasses two LANs, you can direct hosts on one LAN to **lanA-profile** and hosts on the other LAN to **lanB-profile**. For information on setting up multiple LAN profiles, see *DSS for AIX Up and Running!*.

Security Namespace Entries: The types of security entries are as follows:

principal This type of entry contains an individual principal.

principal directory This type of entry contains individual principals or one or more principal directories, or both.

group This type of entry contains an individual group.

group directory This type of entry contains individual groups or one or more group directories, or both.

org This type of entry contains an individual organization.

org directory This type of entry contains individual organizations or one or more organization directories, or both.

policy This type of entry contains a Security policy.

When you (or an application) are accessing an entry in the Security part of the namespace, the name of the entry alone provides enough information for the Security Service to work with. For example, the Security server knows that the login name is a principal name that is registered in the Security part of the namespace; /**.:**/*principal_name*, /**...**/*cell_name/principal_name*, and *principal_name* are all valid ways of representing the name you use to log in.

When you use the **dcecp** program, you specify the type of object you will operate on before you operate on it. For example, to change account information associated with the principal **smith**, you specify that you want to operate on a principal, and you then enter the principal name **smith**. The **dcecp** program deals with the following four types of security objects:

- Principals
- Groups
- Organizations
- Accounts

The Chapter 7, "DCE Security Service" section explains how to use the **dcecp** program to display information related to principals, groups, organizations, and accounts.

In addition to objects registered in the Security space, the **dcecp** program operates on all objects in the namespace. To operate on these objects, the **dcecp** program requires the object's fully qualified path name, as shown in the following example:

```
/.:/sec/principal/smith
```

and not simply the following:

```
smith
```

The following parts of the namespace comprise the Security namespace:

- **/.:/sec/principal**
- **/.:/sec/group**
- **/.:/sec/org**
- **/.:/sec/policy**

CDS Namespace Replication Considerations: Directory replication is the most reliable way to back up the information in your CDS namespace. Because the CDS data is replicated by directory, when you replicate a directory, all of the entries in it are automatically replicated. Use the **dcecp** program to create replicas of directories at a CDS clearinghouse. Clearinghouses need to be created in the root directory (/**.:**) of the cell namespace.

Follow these guidelines for replicating parts of the cell namespace:

- The root directory (/**.:**) is automatically replicated when you create a clearinghouse.
- You should have at least two copies of each CDS directory to ensure that the entire namespace is available at all times. For further information about backing up CDS information, see "Backing Up the Security Service Registry and CDS" on page 3-1.

Planning for Access Control

When planning for access control, it is important to keep the level of access control in your cell restrictive enough to ensure that security is maintained. A special set of individuals or a special group can be given permission to create accounts and groups in the root directory of the Security namespace. A group called **acct-admin** is created when you configure DCE. The **acct-admin** group is the only group that can create accounts and groups in the root directory of the Security space.

While maintaining an adequate level of security in your cell, you also need to consider the requirements of administrators who are maintaining DCE services when you set access control levels. For example, if one person is responsible for administration of DFS in your cell, that person may need to add servers to the Security and CDS namespaces. On the other hand, an administrator responsible for the Security Service manages the Security server but does not control the DFS filespace.

Following are some of the groups created when you configure DCE: (using SMIT or the DCE configuration script)

sec-admin This group administers Security servers, registry replication, and other Security functions.

cds-admin This group administers CDS servers, CDS replication, and other CDS functions.

dts-admin This group administers DTS servers and related DTS functions.

dfs-admin This group administers DFS File Servers and related DFS functions.

audit-admin This group administers the Audit deamon and related Audit Service functions.

In addition to the administrative groups, individual users need permission to control some information kept in the registry database. For example, a user needs to be able to change its password, home directory, or login shell.

Client and Server Considerations

This section describes configurations for DCE client machines, the different types of DCE server machines, DCE remote administration machines and DCE Application Development Environment machines. A DCE Client machine can run client code of every DCE service. DCE server machines are configured to run a certain set of the DCE software. A DCE server software package is made up of at least one daemon and, in some cases, one or more additional programs that comprise the server side of the DCE component. DCE server machines also run the DCE client software. DCE remote administration machines, which are client machines specially configured for remote server administration, contain certain administration programs in addition to the DCE client software. The DCE Application Development Environment configuration contains files (such as header files) needed by DCE application programmers, in addition to the DCE client software.

When planning the configuration of your DCE clients and servers, remember space needs. A machine that has a particular configuration of the DCE software will need enough space for both the DCE software and the operating system software. See *DSS for AIX Up and Running!* for detailed information on space requirements for the various DCE machine configurations.

The following sections are presented in the order in which you need to approach DCE machine configuration.

Requirements for DCE Client Machines

The following topics describe the requirements for setting up DCE client machines. They also discuss some considerations for configuring DCE clients. Remember that all DCE machines, including DCE server machines, are DCE clients. Therefore, be sure to add the appropriate server space requirements to the DCE client space requirements to reach an approximate total space requirement for the machine. See *DSS for AIX Up and Running!* for space requirements.

The following sections describe the executables that run on a DCE Client machine.

RPC Client Programs: A DCE Client contains the following RPC programs:

- The **dced** daemon must run on any machine that has a DCE RPC server process that exports an interface with dynamic bindings. The **dced** daemon is used to register binding information (among other things). The **dced** daemon must run on every DCE machine because on every DCE machine there are client-side daemons that export interfaces. For example, the **dtsd** daemon exports the **acl** interface.

  The **dced** daemon must be running before you configure any other DCE services because DCE services need to register their endpoints with **dced**. Only one **dced** daemon is needed on a machine. In fact, only one can run on a machine at a time because it uses a well-known port.

  Network interfaces, routing services, and other network services must be available before DCE RPC starts. The **dced** daemon is started in the **/etc/rc.dce** file. The **/etc/rc.dce** file can be invoked by other **rc** files, such as **/etc/rc** and **inittab**, so that DCE services can be brought up each time the machine boots.
- The DCE control program (**dcecp**) is used to manage and maintain the DCE RPC software. In addition, the **rpccp** program is used for operating on local registries.

Security Service Client Programs: Every DCE client machine has, as part of the **dced** daemon, the Security Validation Service. This service takes the place of the machine principal. Most principals are interactive users, but the machine principal is not. The Security Validation Service performs the processing necessary so that other daemon processes on the machine appear to be running with the machine's identity.

The Security Validation Service periodically refreshes the ticket-granting ticket for the machine's principal. A DCE client machine must have a valid ticket-granting ticket in order for a principal to use DCE services. The Security Validation Service also exports the interface that assures a Security client that it is actually contacting the real Security server when the client requests a ticket-granting ticket from the Security server.

Audit Service Client Programs: There are no Audit service client programs. The clients of this service are the server processes of the DCE services that use auditing, for example, the Security Service's **secd** daemon.

CDS Client Programs: The DCE Client runs the following CDS processes:

- The CDS advertiser, the **cdsadv** process, does the following:
  - Allows applications to locate and communicate with **cdsd** servers
  - Starts any needed CDS clerks (**cdsclerk**)
  - Creates the cache shared by local CDS clerks
- The **cdsclerk** is an interface between CDS client applications and CDS servers. A clerk must exist on every machine that runs a CDS client application. One **cdsclerk** process runs for each DCE principal on a machine that accesses CDS. The CDS clerk handles requests from client applications to a server and caches the results returned by the server. Because the results of the server request are cached, the clerk does not have to go repeatedly to the server for the same information. All CDS clerks on a machine share one cache. One clerk can serve multiple client applications running on the same machine.

DTS Client Programs: The DCE Client runs the following DTS processes:

- The **dtsd** daemon is set to be a client or a server. On a client machine, **dtsd** synchronizes the local clock.
- The DCE control program (**dcecp**) is for the management of the DTS software. Optionally, so is the **dtscp** program.

DFS Client Programs: If DFS is installed, the DCE Client runs the following processes:

- The Cache Manager process (**dfsd**) initializes the Cache Manager, alters configuration settings, and starts background daemons.

  The **dfsd** process is responsible for the local caching of file and directory data on machines used as DFS clients. When the **dfsd** process starts, it initializes the cache. When a client retrieves part of a file from a remote File Server, the **dfsd** process keeps a copy of that part of the file on the client machine's local disk. As long as that part of the file does not change, the locally cached copy remains available to the client. A new copy is retrieved from the DFS File Server machine only when another process changes the cached portion of the file. The **dfsd** process also caches directory and fileset location information.

- The **dfsbind** process:
  - Obtains cell location information from CDS, and
  - Responds to Security Service requests on behalf of the DFS kernel processes by making calls to the Security Server.

Requirements for DCE Server Machines

The following sections describe the considerations involved in setting up DCE server machines.

Files Installed on DCE Server Machines: The following subsections list the files that must be installed on each of the different DCE server machines. Remember that because all DCE servers are also DCE clients, the files described in “Relocating a Clearinghouse” on page 5-57 must also be installed on server machines. Therefore, add the appropriate client space requirements to the DCE server machine space requirements to reach an approximate total space requirement for the server machine.

DCE RPC Server Programs: There are no DCE RPC server programs other than the programs that run on the DCE Client.

Security Server Processes: Every cell has one master replica Security Service machine and can also have slave replica Security Service machines. The following processes run on a Security Service master or slave replica server machine:

- The Security Server, or **secd** process, implements the Authentication service, the Privilege service, and the Registry service.
- The **sec_create_db** program initializes the security database. You give this command an option indicating whether you want to create a master or slave replica Security server on the machine.
- The DCE control program (**dcecp**) supports the management and maintenance of the Security software. Optionally, so does the **sec_admin** program. See “**sec_admin** Program Usage” on page 7-136 for more information.

Keep the following considerations in mind when you are planning for Security servers:

- The node that runs the master Security server must be highly available and physically secure. Consider placing the master Security server machine in a locked room and keeping a log to record who accesses the machine.
- Be sure to move the master Security server before removing the node from the network or shutting down the node for an extended period of time. Modifications are made to the master security server and propagated to slaves throughout your cell. If the master security server is unavailable, no updates can be made.

- When configuring a Security replica in a cell, ensure that the master Security server has been configured with a CDS client. Otherwise, the Security replica will fail, issuing an error message indicating “One or more ACL modifications failed.”
- A cell can have only one master Security server. If you plan to make one cell out of several existing cells with independent master security servers, you must first merge their registries.
- If the host that contains the master Security server goes down, hosts that have slave servers can still provide registry information, so consider having a number of slaves in your network. Use factors such as the number of machines in your cell, the reliability of the machines that run Security servers, and your cell's available resources to determine how many slave Security servers you need to have.

For further information about planning for the Security Service, see “Planning Sites for Security Service Components” on page 7-103.

Audit Server Processes: An Audit server provides the other DCE services with access to the DCE auditing facilities. An Audit server runs the **auditd** daemon. When auditing is available in a DCE cell, each machine must run the daemon.

CDS and GDA Server Processes: A CDS server stores and maintains object names within a cell and handles requests to create, modify, and look up data. One of the CDS server machines in a cell must be configured as a GDA server as well. There must be a GDA server (the **gdad** daemon) in a cell in order for the cell to communicate with other cells.

The following processes run on a CDS server machine:

- The CDS daemon, **cdsd**, is the CDS server process.
- The **cdsadv**, on a DCE client machine, receives server advertisements to find out what servers are available. On a CDS server machine, it also sends server advertisements.
- The DCE control program (**dcecp**) supports the management and maintenance of the CDS software. In addition, the **cdscp** program is used for controlling and displaying information about CDS clerks and servers. See “**cdscp** Program Usage” on page 5-10 for more information.

When preparing for CDS, you need to select server nodes that store and maintain the clearinghouses (CDS databases) in the cell. Keep the following guidelines in mind in order to achieve reliability, optimum performance, and data availability:

- Choose dependable nodes. A CDS server wants to avoid downtime as much as possible and needs to be restarted quickly when downtime occurs. The CDS server needs to be one of the first systems available on the network because client applications and other DCE servers rely on the CDS server for up-to-date information. The CDS server initializes the CDS namespace when you configure DCE.
- Use reliable network connections. This helps to ensure that all servers maintaining directory replicas can be reached when CDS performs a skulk. Skulks are periodic updates that check for consistency across all replicas.
- Consider the size of your cell and how geographically dispersed the cell is when deciding how many CDS servers you need. You should have at least two copies (one master and one read-only replica) of each CDS directory to ensure access to data if one of the servers becomes unavailable.
- Each CDS server in a cell must maintain at least one clearinghouse. All clearinghouses contain a copy of the root, in addition to other directories replicated there.
- Make replication decisions based on where the contents of directories are referenced. Put read-only replicas where the contents are read and put master replicas where the contents are written.

In a DCE configuration that uses GDS or DNS, CDS must be able to contact at least one GDA to access a global directory service. CDS contacts the GDA via the **gdad** daemon, which sends lookup requests for

cell names to either GDS or DNS and returns the results to the CDS clerk in the cell that initiated the request.

The GDA can be on the same machine as a CDS server, or it can exist independently on another machine. You should have at least two **gdad** daemons running in a cell to ensure GDA availability.

DTS Server Programs: The DCE client configuration already contains all the files necessary for a DTS server machine, with the exception of the optional time provider. The necessary files are as follows:

- The **dtsd** daemon, which is also installed on a DCE client machine, is configured to run as a server when installed on a DTS server machine. As a server process, **dtsd** synchronizes with other DTS servers, in addition to synchronizing the local clock, as it does on a client machine.
- The **dts**_*device_name*_**provider** specifies the communications between the DTS server process and the time-provider process. For *device_name*, substitute the device you are using, which can be a radio, clock, or modem, or another source of UTC time for DTS. A time provider is optional. If you use a time provider, it must connect to a server process.

Consider the following guidelines when planning your DTS implementation:

- Each cell needs at least three DTS servers. At least three DTS servers are needed in order to detect if one of them is faulty when they are queried for the time. It is preferable to have four or more DTS servers to provide redundancy. The additional servers increase the accuracy of time synchronization. However, increasing the number of servers queried for the time also increases the activity on the network. The administrator must balance the level of accuracy with the amount of network activity.
- A time provider is optional in DTS; however, cells that must be closely synchronized with a time standard need to have at least one time provider.
- Servers need to be located at the sites with the greatest number of different network connections.

There are many network configuration decisions that affect DTS planning. “Planning Your DTS Implementation” on page 1-43 contains details about the total DTS planning process, including configuration planning for Local Area Networks (LANs), extended LANs, and Wide Area Networks (WANs). “Criteria for Selecting a Time Source” on page A-2 also explains the criteria you need to use when selecting a time source for your network to use.

DFS Server Programs: DCE supports configuration of the following types of DFS server machines:

- DFS-private File Server machine
- System Control machine
- File Server machine
- Fileset Location Database (FLDB) machine
- DFS Backup Server machine
- DFS Fileset Replication Server machine

DCE Administration Utilities

The following sections describe the utility programs that DCE provides for managing and maintaining the DCE software. The last section tells you which utilities to place on a machine that is specially configured for the remote administration of DCE servers.

DCE Control Program: The **dcecp** program, which is the overall administration tool for DCE, has functions for administering the DCE services except that it does not perform certain operations on CDS clerks and servers and does not maintain local registries. Also, you cannot use the program to administer DFS.

DCE RPC Administration Programs: The **rpccp** program, which is the RPC-specific administrative tool, allows you to browse, update, add, and delete the DCE RPC attributes of entries stored in the CDS namespace and the endpoints that are managed by local and remote **dced** daemons.

Security Service Administration Programs: The Security Service provides the following administration utilities:

- The configuration process, in addition to other functions, creates the initial Security database and sets up some configuration files. See *DSS for AIX Up and Running!* for detailed information on DCE configuration.
- The **sec_create_db** utility recreates a registry from scratch.
- The **acl_edit** command displays, adds, modifies, and deletes ACL entries for a specific object.
- The **rgy_edit** command allows you to edit the Security database or the local registry.
- The **passwd_import** command allows you to create registry entries based on the group and password files from machines that do not implement DCE Security.
- The **passwd_export** command allows you to update the AIX **/etc/passwd** and **/etc/group** files with current user information obtained from the registry.
- The **passwd_override** file allows you to establish overrides to the information contained in the registry.
- The **sec_admin** command helps you manage server replicas of the registry, change the master server site, and reinitialize a slave server.

CDS Administration Programs: CDS provides the following administration utilities:

- The CDS control program (**cdscp**), which is a CDS-specific administrative tool that allows you to control CDS servers and clerks and to manage the namespace and its contents.
- The CDS Browser (**cdsbrowser**) is a Motif-based program that lets you view the contents and structure of the CDS namespace.

DTS Administration Programs: The DTS control program (**dtscp**) allows you to administer DTS, including configuring the **dtsd** daemon as either a client or a server. The **dtscp** program is included in the DCE client software.

DFS Administration Programs: DFS provides utilties to ensure internal consistency, manage filesets, and perform backups. See *DCE for AIX DFS Administration Guide and Reference* for a detailed description of DFS utilities.

Programs for DCE Remote Administration Machines: A system user can perform administrative tasks from any machine in the cell when logged in as the cell administrator. In order to perform administrative tasks, the user must authenticate and ensure that the tools necessary to perform the tasks are available. The administration utilities that need to be installed, in addition to the DCE client software, are:

- The **cdscp** program for controlling certain operations on CDS clerks and servers.
- The **rgy_edit** program for maintaining local copies of the Registry.
- The **passwd_override**, **sec_salvage_db**, **sec_admin** and **sec_create_db** programs must be installed for Security service administration.

No software other than the **dcecp** program and the DCE client software is needed for administering DCE RPC or DTS.

Application Development Environment Machine: A DCE machine can also be configured for the development of DCE applications. This involves adding to the basic DCE client configuration several include (**.h**) and interface specification (**.idl**) files, along with the **idl** program.

Overview of DCE Maintenance

After performing the tasks required for planning, installing, and configuring your DCE system, you can continue with the tasks required for maintaining the system. The initial tasks of planning, installing, and configuring are performed infrequently, some only once. Maintenance tasks, however, are performed on a regular basis throughout the lifetime of your system.

Maintenance of a distributed system includes the following areas:

- Performance tuning
- Configuration control
- Security and access control

The following describes tasks that apply to the individual components of DCE. DCE component tasks are documented in detail in this guide and in the *DCE for AIX DFS Administration Guide and Reference*.

For information about starting or stopping DCE, or changing the network address of a DCE machine, see *DSS for AIX Up and Running!*. See *DCE for AIX DFS Administration Guide and Reference* for a detailed description of DFS maintenance tasks.

Changing the Network Address of a DCE Machine

Occasionally, you will need to change the network address of a machine running DCE. DCE stores the network address in several files that must be updated.

CDS Maintenance Tasks

CDS components, including clerks, servers, and clearinghouses, are largely self-regulating. Except for routine monitoring, CDS requires little intervention for system administration. When intervention is required, CDS provides system administration tools to help you monitor and manage the CDS namespace and CDS servers.

You can use the DCE control program (**dcecp**) commands described in “**dcecp** Commands Usage” on page 2-19 to create and manage the components of a CDS namespace.

You can also manage CDS by using the CDS Browser utility (**cdsbrowser**) to view the namespace. The **cdsbrowser** utility enables you to monitor growth in the size and number of CDS directories in your namespace. You can use the **cdsbrowser** utility to display an overall directory structure, as well as the contents of directories. “Displaying the Namespace with the CDS Browser” on page 5-28 discusses the **cdsbrowser** utility.

If you have a large organization, you can improve efficiency by having one system administrator responsible for CDS servers and another system administrator responsible for the namespace. You can delegate responsibility for a subtree of the namespace to another administrator by granting access control rights to that person.

For more detailed information on CDS maintenance tasks, see the Chapter 5, “Cell Directory Service (CDS)” section.

Monitoring CDS: CDS monitoring tasks fall into the following two categories:

- Monitoring the namespace
  - Monitor the size and usage of clearinghouses and determine the need for new CDS servers and clearinghouses. Plan and oversee the configuration of these new servers and clearinghouses.
  - Maintain and monitor a map of the namespace.
- Monitoring CDS servers
  - Enable event logging, monitor CDS events, and solve system-specific problems if they arise. If necessary, notify the namespace administrator of problems that can affect other CDS servers or clerks.
  - Monitor the success of skulks that originate at the server. A skulk is a method of updating all replicas through repeated operations.
  - Monitor the size and usage of the server's clearinghouse and, if necessary, discuss with the namespace administrator the need to relocate some replicas or create a new clearinghouse.
  - Monitor and tune system parameters that affect or are affected by CDS server operation.

Note: When monitoring memory usage for CDS servers, it is important to understand that memory remains allocated under certain conditions. Memory associated with objects remains allocated until a skulk is successfully completed. Memory associated with directories remains allocated until the server is disabled and restarted.

Managing CDS: CDS management tasks fall into the following two categories:

- Managing the namespace
  - Oversee the creation of new directories and assign names according to a standard, or enforce established guidelines in the assigning of names. (Beyond a certain level in the directory hierarchy, you can delegate the responsibility of creating and maintaining directories. You need to keep track of the new directories being created to make sure they are appropriately replicated.)
  - Determine the default access control policy.
  - Administer and enforce the established access control policy for directories and entries.
  - Determine where and when new replicas of a directory are necessary.
  - Create soft links for objects that change locations or for objects that need to be renamed. An object is a resource, such as a disk, an application, or a node, that is given a CDS name. A name plus its attributes make up an object entry. A soft link is a pointer that provides an alternate name for an object entry.

    Publicize and encourage the use of the new names so that eventually the soft links can be deleted.
  - Solve or direct the resolution of problems involving multiple CDS servers.
- Managing CDS servers
  - Manage access control on directories and objects, and monitor the size and usage of directories in the server's clearinghouse. Create new directories, possibly with the namespace administrator, when necessary.
  - Create new objects in directories or oversee their creation. (Beyond a certain level in the directory hierarchy, you also can delegate the responsibility of maintaining directories and the objects in them.)
  - Add new administrators to the **cds-admin** security group.

CDS Security and Access Control: The DCE control program (**dcecp**) and CDS ACL Manager work together to manage authorization in CDS. To modify, add, delete, or view ACL entries in the CDS namespace, use the **dcecp** program's **acl** commands. When the **dcecp** program issues a request to perform an operation on a CDS object, the CDS ACL Manager checks permissions, based on ACL entries, and grants or denies the request. The CDS ACL Manager is an integral part of the **cdsd** and **cdsadv** processes.

The Chapter 5, "Cell Directory Service (CDS)" section provides detailed information about handling CDS security and access control, including quidelines for setting up access control in a new namespace.

DTS Maintenance Tasks

Like CDS, DTS is largely self-regulating after configuration of the service is complete. However, there are times when you need to intervene. Use the **dcecp** program to perform the following DTS configuration and management tasks:

- Identify system clock problems
- Adjust the system clocks
- Change DTS attributes to varying WAN conditions
- Modify system configuration when the network environment changes

For more information on DTS maintenance tasks, see the Chapter 6, "DCE Distributed Time Service" section.

Managing the Distributed Time Service: You can use the **dcecp** program interface to create and enable DTS. After this is done, you can perform routine management tasks, such as enhancing performance, reconfiguring the network, and changing local time.

Several commands and characteristics modify and improve the performance of your network. The **dts modify** command changes the values of many of these characteristics. The **dts show** command displays the values of characteristics at any time. The following are some of the tasks you can accomplish using the DTS commands and the characteristics of DTS that can be set:

- Display or change the number of servers that must supply time values to the system before DTS can synchronize the system clock
- Display or change the inaccuracy limit that forces the system to synchronize in order to bring the inaccuracy back to an acceptable level
- Display or change the interval at which you want clock synchronization to occur
- Display or change the reaction to a faulty system clock
- Display or change the settings that indicate how often to query servers.
- Creating and enabling DTS.
- Assigning the courier role to servers to facilitate communication to other parts of your network.
- Matching the epoch number for servers you add to your network after initial configuration. An epoch number is 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.
- Advertising DTS servers to CDS, thereby registering them as objects in the namespace.

Modifying System Time: Sometimes you need to modify the system time. You can update time to match the international time standard, Coordinated Universal Time (UTC), from a source such as telephone, radio, satellite, or another external referencer, if your network does not use time providers and network systems have been running for some time. The **clock set** command accomplishes this task by gradually modifying the time.

The **clock set** command used with the **-abruptly** option and the **dts synchronize** commands provide additional methods for adjusting the system clock and synchronizing systems.

Note: DCE for AIX supports the use of external time providers by making example code available. However, neither the example code nor the external time providers are supported by IBM.

Security Service Maintenance Tasks

The following sections summarize the maintenance tasks you perform while administering the Security Service. For more detailed information on Security maintenance tasks, see Chapter 7, “DCE Security Service” on page 7-1.

Managing the Security Service: The Security Service management tasks include the following:

- **Creating and maintaining accounts by using the dcecp program** — The **dcecp** program provides commands for creating and maintaining registry information, including persons, groups of users, and accounts.

  Keep the following things in mind when administering DCE accounts:

  – If you share files with other systems that do not use the registry, be sure that names, UNIX IDs, and account information are consistent between the registry and the foreign password and group files. Use **passwd_import** to identify and resolve any conflicts that exist. “**passwd_import** Operation” on page 7-100 describes how **passwd_import** works.

  – If you maintain **/etc/passwd** and **/etc/group** files in standard UNIX format, you need to run **passwd_export** to make password, group, and organization files on local machines consistent with the registry. See “Ensuring Consistent Local Files” on page 7-74 for more information about the **passwd_export** command.

  – For principals in other cells to access objects in your cell, you need to set up a special account for the foreign cell in your cell's registry. This account indicates that you trust the Authentication Service in the foreign cell to correctly authenticate its users. Use the **dcecp** program's **registry connect** command to create an account for a foreign cell.

- **Using ACLs** — Use the **dcecp** program's **acl** commands to display, add, modify, and delete ACL entries for a specific object in the cell namespace. See the “ACL Entries and Masks” on page 7-1 for detailed information on how to use the **dcecp** program's **acl** command.

- **Setting and maintaining registry policies** — Registry policies include certain password and account information. Policies also include overrides, which are exceptions tied to a specific machine. Use the **dcecp** program's **registry** commands to set and maintain registry policies. Details on how to use these commands are in “Maintaining Policies and Properties” on page 1-58.

  Ticket expiration date, password lifespan, password format, and password expiration date are examples of registry policies that you can set. If both an organizational policy and a registry policy exist for password format, for example, the more restrictive policy applies.

  You can establish overrides to the information contained in the registry. Override information is stored in the **passwd_override** and **group_override** files on a local machine. The **passwd_override** file contains the home directory, the login shell, entries for overriding the password, and GECOS information, which is general information that is used by users but not required by the system, such as office and phone numbers. For details about how to edit the **passwd_override** file, see “Overriding Entries in the Local /etc/passwd and /etc/group Files” on page 7-95.

- **Backing up the registry** — “Backing Up and Restoring the Registry Database” on page 7-36 describes the back-up procedure to follow for the primary registry site. When you restore the database, it is automatically propagated to the secondaries.

- **Setting up and maintaining Audit Service data** — Audit Service data includes event numbers, event class numbers, event class files, audit filters, and audit trail files. Use the **dcecp aud, audevents, audfilter**, and **audtrail** objects to manage Audit Service data. The *DCE for AIX Administration Command Reference* provides descriptions of audit-related **dcecp** objects and commands. See "DCE Audit Service Administrative Tasks" on page 1-49 for more information.
- **Troubleshooting** — When you encounter problems that cannot be resolved through routine management procedures, or when hardware failures stop the registry from operating, there are several troubleshooting procedures you can use. This guide describes the following tasks:
  - Recreating a registry replica
  - Recovering the primary registry
  - Forcibly deleting a replica
  - Adopting registry objects that are orphaned because their owner has been deleted

Reconfiguring the Registry: There are two main reconfiguration tasks included in the administration of the Security Service. The following tasks are described in "Removing a Server Machine from the Network" on page 7-125 and "Changing the Master Replica Site" on page 7-37.

- Changing the primary registry site when you plan to move the machine that runs the primary registry server from your network or shut the machine down for an extended period
- Removing a server host from the network if you plan to remove a machine that runs a secondary registry server from the network or shut that machine down for an extended period

Removing Expired Credentials Files: The /**var** file system may periodically fill up, especially if expired credentials files are not removed. If you are root, you can use the **rmxcred** tool to remove expired tickets from the credentials directory (/**opt**/**dcelocal**/**var**/**security**/**creds**). You should invoke this tool on a regular basis, such as setting up an AIX **cron** job to run the tool daily.

DCE Control Program Overview

DCE is an integrated set of services that supports the development and execution of distributed applications between heterogeneous networked computers. Each DCE environment (called a cell) maintains at least the following core DCE services:

- DCE Threads
- DCE Host Services
- DCE Cell Directory Service
- DCE Time Service
- DCE Security Service

With the exception of DCE Threads, all of the core services require administration in one way or another. Some services like CDS and DCE Security usually need more managing than say, the DCE Time Service which, after you have set it up, needs practically no intervention.

If your DCE cell consists of just a few computers and their users, you could probably manage the naming, time, and security needs of users, programs, and host systems by logging onto individual hosts to perform any necessary administration tasks. But most cells will consist of many, perhaps hundreds or even thousands of computers and their users. Consequently, the core services in these cells will likely be large and complex with some services being replicated or even partitioned across multiple heterogeneous systems. Some services, such as the DCE host services, will exist on every computer in the cell. Such large scale operations demand an administrative interface that provides consistent and uniform access to DCE administration functions, wherever they reside, from any and every point in the cell. This means that

administrative operations must work consistently and predictably regardless of the platform on which they execute.

The DCE Control Program (**dcecp**) available with DCE fills this need, providing consistent, portable, extensible, and secure access to nearly all DCE administration functions from any point in a DCE cell. The **dcecp** program implements most of the operations previously performed by using various component control programs.

The **dcecp** program further streamlines administration by providing a number of task objects for performing complex DCE operations. For example, adding a host to a cell requires adding a host principal to the registry, adding the principal to various security groups and organizations, creating an account, placing host information in CDS and probably setting some ACLs on CDS directories. All of these operations can be accomplished using a single task object.

For additional information, see:

- "File Locations" on page A-3
- "DCE Administration Objects" on page 2-15
- "DCE Control Program Usage" on page 2-15
- "**dcecp** Commands Usage" on page 2-19
- "Interactive Command or Script Usage" on page 2-40
- "Editing Command Lines" on page 2-24
- "**dcecp** Help Facilities Usage" on page 2-21
- "Customizing **dcecp** Sessions" on page 2-13

Command Language Overview

The DCE control program is built on a portable command language called Tcl (pronounced *tickle*), which stands for Tool Command Language developed by John K. Osterhout at the University of California at Berkeley, California. Most computers provide a command language of some sort to give users a flexible and extensible way to access and use system capabilities. For instance, many UNIX-style systems offer shell language interpreters and Digital Equipment Corporation's OpenVMS operating system offers the Digital Command Language (DCL). But these command languages are not always portable. Commands and scripts based on one command language might not work in other command language environments.

Tcl, on the other hand, is a platform-independent command language that runs on every system where OSF DCE Version 1.1 is installed. A Tcl command interpreter and the DCE Control Program which uses it are provided as part of the OSF DCE Version 1.1 software, which is the basis for the DCE for AIX 2.1 product.

Note: The DCE implementation of the Tcl command interpreter is called the DCE control program language. From now on we will refer just to the DCE control program language. The Tcl command interpreter is designed expressly to support the DCE control program language. You will likely have difficulty if you try to execute generic (non-DCE) Tcl scripts using the **dcecp** command interpreter.

The availability of both the DCE control program and the DCE control program language offer important benefits to DCE administrators:

- You can perform virtually all routine DCE operations from within a single administrative interface.
- Most DCE administrative operations are consistently and uniformly executed from any OSF DCE Version 1.1 platform allowing administrators to manage just about all DCE operations from any OSF DCE Version 1.1 system in the cell. Non-UNIX OSF DCE Version 1.1 platforms might not handle all DCE control program file operations.

- The **dcecp** program provides administration objects with names like **clearinghouse**, **principal**, and **endpoint**. This direct approach makes DCE administration intuitive and consistent. While for now, this has only the appearance of being object oriented it is an important step toward a true object-oriented administration interface.
- Task objects (high-level **dcecp** scripts that perform complex DCE operations) reduce the training requirements for DCE administrators. One need not be a DCE guru to perform routine DCE administrative tasks.
- You can adapt the supplied task objects to new uses or write new task objects or scripts using the **dcecp** operations along with more general commands provided within Tcl.
- The **dcecp** language allows the use of variables, **if** statements, looping functions and other programming operations that let you boost the power of your operations. For instance, looping functions let you repeat operations on multiple objects such as users, servers, or CDS entries.
- Administrators can easily share their tools because scripts can be moved to foreign platforms without change. For instance, enterprises with multiple cells could use **dcecp** scripts to propagate a common cell configuration throughout the enterprise.

The DCE control program is an administrative interface that you can use to manage most aspects of the DCE core components. You cannot use **dcecp** to manage every aspect of DCE. First, not all of the existing programs are replaced by **dcecp**. For instance **dcecp** cannot control GDS or DFS. Second, even when **dcecp** replaces a control program such as **cdscp**, not every operation in the control program is necessarily implemented by **dcecp**. Typically unimplemented operations are those that are performed only once or relatively infrequently. OSF plans to implement all DCE operations in **dcecp** eventually. For now, you will have to use the individual component control programs to perform operations not implemented in **dcecp**.

The tasks described in Chapter 2, “The DCE Control Program” on page 2-1 illustrate the use of the **dcecp** program to administer the core services in your DCE environment. We also discuss how to make your operations do more by using Tcl constructs on the command line and by writing your own customized operations as scripts. We do not provide a complete discussion of Tcl or its companion toolkit (called Tk) for the X11 window system. For in-depth discussions of these topics, refer to *Tcl and the Tk Toolkit*, John K. Osterhout, (c)1994, Addison Wesley Publishing Company.

DCE Control Program Language Usage

In “**dcecp** Commands Usage” on page 2-19, we provide a high-level look at some ways to use the DCE control program to administer your DCE environment. In this section, we discuss some syntax rules and some of the more important commands you will need to use in composing your **dcecp** administration commands and task scripts.

The **dcecp** command language consists of DCE administration commands like **directory create** and **object modify**, as well as Tcl built-in commands such as **if** and **foreach**. We will not discuss DCE administration commands here. These commands are discussed in sections that deal with administering the particular DCE component. Instead, we will focus on using the more generic syntax rules and built-in commands.

The Tool Command Language (Tcl) on which **dcecp** is based is a general-purpose language that is also used for other applications besides **dcecp**. Although there are many ways you can use Tcl for various purposes, we will limit our discussion to those commands most likely to be used for administering DCE environments. Furthermore, our command discussions do not describe every aspect of individual commands. Rather they suggest why and how you might use a command in the context of administering a DCE environment. If you are not already familiar with Tcl, you will likely need to have access to the

appropriate Tcl documentation, including the Tcl reference pages, for writing sophisticated commands and task scripts.

For additional information, see:

- “Variable Substitution” on page 2-48
- “Command Substitution Usage” on page 2-1
- “Grouping Elements and Controlling Interpretation” on page 2-37
- “Documenting Your Scripts with Comments” on page 2-23
- “Convenience Variable Usage” on page 2-5
- “Measuring and Counting with Expressions” on page 2-44
- “Operating on Lists” on page 2-45
- “Controlling Your Script” on page 2-1
- “Creating Commands Dynamically with the **eval** Command” on page 2-10
- “Reading Other Files as **dcecp** Scripts with **source** Command” on page 2-46
- “Creating New Commands With the **proc** Command” on page 2-10
- “Manipulating Strings” on page 2-41
- “Error and Exception Handling” on page 2-26
- “Working with Files” on page 2-49
- “Spawning Subprocesses” on page 2-47

Writing Scripts and dcecp Objects

The DCE control program supplies a number of objects that offer administrative access to each manageable component in a DCE cell. For instance, the **principal** object lets administrators manage principal information in the DCE Security Service registry database. Similarly, the **rpcgroup** object lets administrators manage group information in CDS.

Some DCE operations affect multiple components as when several operations must be performed to add a new user to a DCE cell. To meet this need, the DCE control program provides task objects which let administrators operate on multiple components with a single operation. For instance, the **user** task object performs several operations that include creating principal information in the registry, adding the principal to an organization and to relevant groups, creating a CDS directory for the user, and so on. Task objects look and behave just like other **dcecp** objects, implementing the same help system used by other **dcecp** objects. However, task objects are written using the **dcecp** language instead of the C programming language. This makes it easy for administrators to extend or customize existing scripts.

While the DCE control program provides task objects to handle some multicomponent operations, variations in cell configurations and differences in the ways administrators manage their cells make it impractical for the supplied DCE task objects to satisfy all the needs of every DCE cell. For instance, some cells may use DFS or GDS components or a cell may implement a cell directory naming scheme that differs from the standard OSF DCE implementation. Alternatively, some DCE implementations could have specialized administrative components such as services or repositories that need distinct **dcecp** objects for managing them.

To accommodate a cell's specific needs, the DCE control program language lets administrators create their own scripts. Administrators can also extend or modify existing task objects or they can create new task objects to manage specialized components in a DCE cell.

For additional information, see:

- “Informal Administration Scripts” on page 2-39
- “Formal Task Objects” on page 2-29

DCE Administrative Tasks Overview

This section provides an introduction to task objects and procedures for managing cells, hosts, users, and cell names.

DCE Administration Task Objects

Often, a single task object uses or affects multiple DCE services. For example, one of the task objects, the **host** object, can configure a host computer into a DCE cell. This task adds specific kinds of information to the DCE Security Service, the Cell Directory Service, and the DCE host daemon services. Because a single invocation of the **host** object can perform multiple steps, it shields DCE administrators from some of the lower-level administration details that would otherwise have to be attended to by using several lower-level **dcecp** administration objects.

For additional information, see:

- "Using Task Objects to Simplify DCE Administration" on page 3-22
- "Looking Beyond the Tools" on page 2-41

Managing a DCE Cell

From a cell administrator's point of view, a DCE cell consists of a set of networked services that supports the execution of distributed applications. This simple statement, however, does not really say anything about what services are currently available in your cell. In fact, the exact number of DCE servers and their locations differs from cell to cell. Even in the same cell, host and network outages and reconfigurations affect service availability.

Although you could use various service-related **dcecp** objects to test whether and where services are available in a cell, it would be cumbersome. Instead, the DCE control program provides a **cell** task object that conveniently lists configured DCE servers and tests whether services are available. It can also back up critical data maintained by the DCE Security Service and CDS.

For additional information, see:

- "Displaying All Configured DCE Servers and DCE Hosts" on page 3-10
- "Testing Cell Operation" on page 3-21
- "Backing Up the Security Service Registry and CDS" on page 3-1
- "Modifying or Extending the Cell Object" on page 3-13

Managing DCE Hosts

Larger DCE cells can contain many host computers, with some running both DCE servers and application servers while others act only as client systems. Still other hosts might run application servers but also act as clients to their resident users. Such flexibility in DCE host configurations can make it difficult to control or track what's running or available on each host in a cell. The **host** task object represents DCE and application processes associated with hosts, letting administrators more easily manage DCE server and application processes on machines.

You can use the **host** task object to show information about processes on local and remote hosts in a cell, and start and stop DCE processes on hosts throughout a cell. You can also configure local DCE hosts in a cell and remove (unconfigure) remote DCE hosts from a cell. Online help for this object is available using the **host help** and **host operations** commands in **dcecp**.

All of the **host** object operations performed on a remote host except **host catalog** require **dced** to be running on the remote host.

For additional information, see:

- “Listing the DCE Hosts in a Cell” on page 3-12
- “Displaying All Servers Configured for a DCE Host” on page 3-11
- “Testing Whether a DCE Host is Running” on page 3-22
- “Starting Configured DCE Processes on a Host” on page 3-20
- “Stopping DCE Processes Running on a Host” on page 3-20
- “Configuring a DCE Host in a Cell” on page 3-3
- “Removing a DCE Host from a Cell” on page 3-14
- “Modifying or Extending the Host Object” on page 3-13

Managing DCE Users

One of the most frequent DCE administration tasks is likely to be managing users in your DCE environment. Corporate reorganizations, changing business needs, and fluctuating economics all exert pressures causing users to come and go or to move between various groups or organizations.

DCE users represent a big part of what DCE is designed to support; the DCE services authenticate and admit some while denying access to those who are unauthorized. Indeed, users have complex management requirements; their information is spread among multiple services that help validate and control their activities. User information includes principal names, group and organization information, account information, and information in CDS.

The DCE control program includes separate administration objects for managing each piece of user information in a DCE cell. While these separate objects might be very useful for making minor adjustments to certain user information, their constant use for repetitive tasks such as adding and removing users from a cell would prove quite tedious. A simpler method relies on the **user** task object that you can use to more easily create, delete, and show user information in a DCE cell. Online help for this object is available using the **user help** and **user operations** commands in **dcecp**.

For additional information, see:

- “Creating a New User” on page 3-3
- “Displaying User Information” on page 3-11
- “Deleting a User” on page 3-9
- “Modifying or Extending the User Object” on page 3-13

DCE Application and Host Administration Overview

This section introduces management of DCE applications, host services, and host data.

DCE Application Administration

As DCE evolves, commonly needed functions are being included in the DCE infrastructure. As an example, DCE Version 1.1 includes server control capabilities that can manage server operation and help servers exit in a controlled and efficient manner. Application developers can rely on these capabilities rather than implement special mechanisms to handle them independently in every server.

Moving commonly needed functions out of applications and into the DCE infrastructure provides important benefits. Applications can be smaller and easier to develop and maintain. Even more important, because applications are not encumbered with lots of special code, they are easier to reconfigure and reconnect

with different kinds of clients. This adaptability is critical as organizations strive to keep up with changing business needs.

DCE applications have always had administrative aspects. Often, programs include the necessary functions to manage their own administrative needs, but this approach can be awkward and somewhat inflexible for administrators. Now, virtually all administrative functions are available to programmers and administrators alike through **dcecp**. This does not mean programmers no longer need to deal with these issues. We expect some programmers to provide scripts written with **dcecp** that configure client and server programs to start and stop under specified conditions. Although this approach offers a convenient and consistent way to administer applications, it also creates an area where programming and administrative concerns overlap.

For additional information, see:

- "Controlling Server Operation" on page 4-3
- "Managing Client/Server Binding Information" on page 4-17
- "Easing Application Development and Administration" on page 4-14
- "Managing Server Entries, Groups, and Profiles in CDS" on page 4-21
- "Client Administration" on page 4-1

Managing DCE Host Services and Host Data

Some services like DTS, CDS, and the DCE Security Service Registry, which produce or maintain cell-wide information, are centralized. Although the services they provide are available throughout a cell, the servers themselves typically reside on just a few selected hosts in a cell.

Other DCE services are pervasive; that is, they reside on every host in a DCE cell. The DCE software that runs on every DCE host provides essential services that enable local client and server programs to interact with remote client and server programs in a reliable and secure way. Consequently, each host in a DCE cell has administrative aspects which are discussed in the first part of this section.

Each DCE host maintains local data that is essential to host operation in a DCE environment. Occasionally, you may find it necessary to modify parts of this data as your cell configuration changes, or as you add DCE capabilities or DCE applications.

When DCE operations do not succeed for some reason, you must inform the right people about what happened. The DCE serviceability messaging facility lets you route error messages based on the severity level of the message.

For additional information, see:

- "DCE Host Services" on page 4-13
- "Managing Host Data" on page 4-18
- "Routing Serviceability Messages" on page 4-33

Cell Directory Service (CDS) Overview

The Cell Directory Service (CDS) is a high-performance distributed service that provides a consistent, location-independent method for naming and using resources inside a cell. CDS offers the ability to replicate CDS names (that is, to store copies of them on more than one node). CDS automatically keeps multiple copies up to date. Names can also be distributed among several nodes so that no one node has to store all of them. This feature is particularly valuable in large cells.

The ability to replicate and distribute information has many benefits, including the following:

- **Availability** — Because you can store the same name in more than one place, data is likely to be available even in the event of a system or network failure.
- **Efficiency** — CDS finds names efficiently because you can store them close to where they are used most often. Furthermore, after CDS finds a name, it can connect to the same name immediately on all subsequent lookups.
- **Load Sharing** — Because names are in more than one place, several systems can share the load of looking them up.
- **Expandability** — New names are easily accommodated as the network grows and more applications use CDS.

For additional information, see:

- "CDS Operations" on page 5-8
- "Replicas Overview" on page 1-31
- "Security in the Cell Directory Environment" on page 5-61
- "Cell Directory Service User Interface" on page 5-11

Controlling Access to CDS Names

The following sections present management information on the authorization components and permissions supported by CDS — controlling access, program commands, and editing ACLs on CDS names.

- "Overview of DCE Authorization for CDS" on page 5-54
- "ACL Types Supported by CDS" on page 5-2
- "Propagating Permissions to CDS Directories and Their Contents" on page 5-56
- "ACL Entry Types Used for Principals" on page 5-1
- "DCE Permissions Supported by CDS" on page 5-20
- "Controlling Access to CDS Clerk and Server Management Operations" on page 5-14
- "CDS Control Program Commands and Required Permissions" on page 5-6
- "Editing ACLs on CDS Names" on page 5-30
- "CDS Servers Access to the Namespace" on page 5-10
- "Setting Up Access Control in a New Namespace" on page 5-62

DCE Directory Service Overview

Distributed processing involves the interaction of multiple systems to do work that is done on one system in a traditional computing environment. One challenge resulting from this network-wide working environment is the need for a universally consistent way to identify and locate people and resources anywhere in the network.

The Distributed Computing Environment (DCE) Directory Service makes it possible to contact people and to use resources such as disks, print queues, and servers anywhere in the network without knowing their physical location. The DCE Directory Service is much like a telephone directory assistance service that provides a phone number when given a person's name. The DCE Directory Service, given the unique name of a person, server, or resource, can return the network address and other information associated with that name.

The Directory Service stores addresses and other relevant information as attributes of the name. For example, attributes can contain the name of an organizational unit (such as European Sales), a location (such as the first floor of Building A), or a telephone number.

For additional information, see:

- "Directory Services Usage by DCE Components" on page 5-25

- “Directory Service Usage” on page 5-27
- “Directory Services and the Cell Environment” on page 5-25
- “Cells Determine Naming Environments” on page 5-12
- “An In-Depth Analysis of DCE Names” on page 5-2

Displaying the Structure and Contents of a Namespace

When you need to view the structure and contents of the cell namespace, you can use one or more programs provided by CDS. The CDS Browser (**cdsbrowser**) allows you to display namespace information in a windowing environment; while the **dcecp** and **cdscp** programs display the information through their command-line interfaces.

For additional information, see:

- “Displaying the Namespace with the CDS Browser” on page 5-28
- “Listing the Contents of Directories” on page 5-37

Looking Up Names With CDS

The following sections illustrate the relationship between a name and the physical resource that it describes, and it explains how CDS handles requests to look up names. Understanding these concepts can help you plan the location of clearinghouses and directories in your cell namespace. It can also help you isolate the source of a problem if you encounter lookup errors or failures.

- “Translating from Names to Resources” on page 5-65
- “Locating Names” on page 5-40

Note: The examples in the referenced sections do not reflect the actual structure of a typical DCE cell namespace. For simplicity, the figures show fewer directories and directory levels.

Managing CDS Directories

If you manage a namespace in a small, slow-growth network of 25 nodes or less, you can maintain all your names in the root directory and may not need to create additional directories. However, if you manage a namespace in a network of more than 25 nodes, you should consider creating at least one additional level of directories under the root.

For additional information, see:

- “Creating Directories” on page 5-15
- “Creating a Read-Only Replica” on page 5-17
- “Deleting Read-Only Replicas” on page 5-24
- “Skulking a Directory” on page 5-64
- “Modifying a Directory's Convergence” on page 5-50

Managing Clerks, Servers, and Clearinghouses

CDS clerks, servers, and clearinghouses are initially created and started as part of the CDS clerk and server configuration. Thereafter, clerk and server processes are created and started with a series of commands that are executed manually or by the start-up scripts on the systems where they are running. These CDS entities are largely self-regulating and, apart from routine monitoring, require only minor management intervention.

For additional information, see:

- “Monitoring Clerk, Server, and Clearinghouse Counters” on page 5-54

- “Monitoring Clerk Communications with Specific Clearinghouses” on page 5-53
- “Displaying the Contents of a Clearinghouse” on page 5-28
- “Forcing the Clearinghouse to Checkpoint to Disk” on page 5-34
- “Disabling Clerks and Servers” on page 5-28
- “Restarting Clerk and Server Processes” on page 5-59
- “Preserving a Clearinghouse Across a Server System Upgrade” on page 5-56
- “Backing Up Namespace Information” on page 5-5

Managing Intercell Naming

To find names outside of the local cell, CDS clerks must have a way to locate directory servers in other cells. The Global Directory Agent (GDA) enables intercell communications by serving as a connection to other cells through the global naming environment.

For additional information, see:

- “Global Directory Agent Operation” on page 5-34
- “Managing the Global Directory Agent” on page 5-45
- “Enabling Other Cells to Find Your Cell” on page 5-31

Managing the DCE Cell Directory Services

The DCE control program (**dcecp**) provides most of the commands you need to manage CDS. A few CDS management tasks cannot be performed using the **dcecp** program. To perform these tasks, you need to use the CDS control program (**cdscp**). The following sections provide some information that you need to run the **cdscp** program and notes the operations for which the program's commands are required.

For additional information, see:

- “Managing CDS with **dcecp** Commands” on page 5-42
- “**cdscp** Program Usage” on page 5-10

For detailed descriptions of the **dcecp** and **cdscp** program commands discussed in this guide, refer to the *DCE for AIX Administration Command Reference*.

Replicas Overview

Directories are the units you use to distribute and replicate names throughout the cell namespace. Each physical copy of a directory, including the original, is called a *replica*. When you create a replica of a directory, you replicate all of the entries in it as well.

Replicas are stored in clearinghouses. You can think of a clearinghouse as the collection of directory replicas at a particular server. After you create a directory in one clearinghouse, you can create replicas of it in other clearinghouses to increase availability for looking up information. CDS periodically ensures that the contents of all replicas of a directory remain consistent.

Two types of replicas can exist — Master and Read-Only. A replica's type affects the processing that can be done on it and the way CDS updates it. The type of replica that CDS uses when it looks up or changes data is invisible to users. However, it helps to understand how the two types differ.

The master replica is the first instance of a specific directory in the namespace. After you make copies of the directory, you can designate a different replica as the master, if necessary. However, only one master replica of each directory can exist at a time. See “Restructuring a Namespace” on page 1-33 for complete information on how to redesignate the master replica of a directory.

The master replica is the only replica of a directory that can be directly modified. CDS can create, change, and delete information in a master replica. Because it can be modified, the master replica incurs more overhead than read-only replicas, which CDS keeps up-to-date periodically with changes made to the master replica.

A read-only replica is a copy of a directory that is available only for looking up information. CDS does not create, modify, or delete names in read-only replicas. It updates them with changes made to the master replica.

Replicas can contain three kinds of entries — object entries, soft links, and child pointers.

Object Entries

An object is any real resource (like a disk, application, or node) that is given a CDS name. When an object name is created, client applications and the CDS software supply attributes to be stored with the name. An attribute, consisting of an attribute name and value(s), describes a particular operational property of an object. The name and its attributes make up the object entry. When a client application requests a lookup of the name, CDS returns the value of the relevant attribute or attributes.

Object entries are typically created and managed through a client application interface. For example, the DCE control program and the Name Service Interface (NSI) of the RPC runtime lets users create entries that represent RPC servers, groups, and profiles. These are special kinds of entries that enable an RPC application to locate and select servers. (See the *DCE for AIX Application Development Guide — Core Components* for details on how RPC uses CDS for this purpose.)

You can also create object entries through the DCE control program (**dcecp**). (See the *DCE for AIX Administration Command Reference* for information on the commands that allow you to create and manage object entries using the **dcecp** program.)

Every object can have a defined class, which is an optional attribute of the object entry. DCE components that use the Directory Service can define their own object classes and supply class-specific attributes for the Directory Service to store on their behalf. Class-specific attributes have meaning only to the particular class of objects with which they are associated.

The clearinghouse object entry represents a special class of object that is predefined by CDS. A clearinghouse object entry serves as a pointer to the location of a clearinghouse in the network. CDS needs this pointer so that it can look up and update data in a clearinghouse.

When you create a clearinghouse, CDS creates its clearinghouse object entry automatically. The clearinghouse object entry acquires the same name as the clearinghouse. The clearinghouse object entry is like any other object entry in that it describes an actual resource, but it is different because it is solely for internal use by CDS. Clearinghouses can be created only in the cell root directory. Therefore, all clearinghouse object entries are stored in the cell root directory. CDS itself updates and manages clearinghouse object entries when necessary. Object entries do not require any external management except in rare problem-solving situations. See your vendor for help in these situations.

Soft Links

A soft link is a pointer that provides an alternate name for an object entry, directory, or other soft link in the cell namespace. You can do minor restructuring of a namespace by creating soft links that point from an existing name to a new name. Soft links can also be a way to give something multiple names so that different kinds of users can refer to a name in a way that makes the most sense to them.

Soft links can be permanent or they can expire after a specified period of time. If the name that a soft link points to is deleted, CDS automatically deletes the soft link when it expires.

Note: CDS managers should use soft links carefully. They should not use soft links to completely redesign the namespace or to provide shortcuts for users who do not want to use the full name of an object entry. Overuse of soft links makes CDS names more difficult to track and manage.

Child Pointers

A child pointer provides a connection between a directory and another directory immediately beneath it in a cell's namespace.

Users and applications do not create child pointers. CDS creates a child pointer automatically when someone creates a new directory. The child pointer is created in the directory that is the parent of (one level above) the directory to which it points. CDS uses child pointers to locate directory replicas when it is trying to find a name. Child pointers do not require management except in rare problem-solving situations.

Summary

To summarize, a cell consists of a complete set of names that are shared and managed by one or more CDS servers in a cell. A name can designate a directory, object entry, soft link, or child pointer. The logical representation of a cell namespace is a hierarchical structure of directories and the names they contain. Every physical instance of a directory is called a *replica*. Names are physically stored in replicas and replicas are stored in clearinghouses. Any node that contains a clearinghouse and runs CDS server software is a CDS server.

Figure 1-3 shows the components of a CDS server node. Every server manages at least one clearinghouse containing directory replicas. A replica can contain object entries, soft links, and child pointers. The figure shows only one replica and one of each type of entry that is possible in a replica. Normally, a clearinghouse contains many replicas and a replica contains many entries.

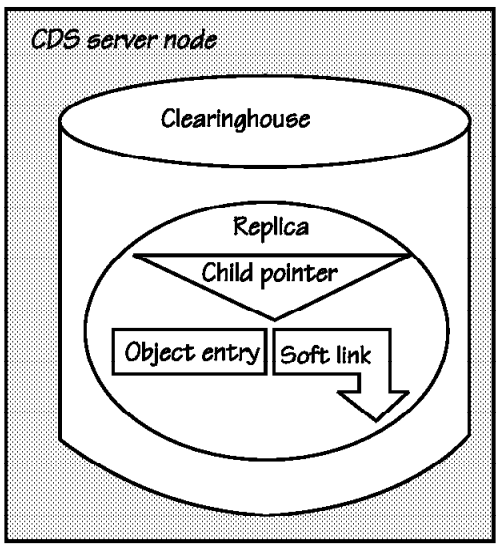


Figure 1-3. Components of a CDS Server Node

Restructuring a Namespace

Over time, you may need to restructure or rename certain elements of your namespace. For example, you may want to create soft links to provide users with one or more alternate names for an existing namespace entry. You may need to reconfigure a directory's replica set to modify the locations and replica types of particular replicas or exclude a replica from the set. Occasionally, you may want to delete certain directories when the information that they contain is no longer needed by users. You may also

need to relocate a clearinghouse or delete a clearinghouse from a server system to perform diagnostic or troubleshooting work on the system, or to prepare for removing the system from your network.

For additional information, see:

- “Managing Soft Links” on page 5-43
- “Modifying a Directory's Replica Set” on page 5-50
- “Deleting Directories” on page 5-22
- “Relocating a Clearinghouse” on page 5-57
- “Deleting a Clearinghouse” on page 5-21

Restructuring CDS Directories in the CDS Namespace

Sometimes, because of corporate restructuring or for other reasons, you need to combine or rearrange various directories or subtree of directories in your CDS namespace.

For example, suppose the engineering group in your organization, /**.:/eng**, is combined with the research and development group, /**.:/rnd**, and that the two groups begin to share a common set of applications and other network resources. You can reflect this organizational change in your namespace hierarchy by merging the contents of these directories.

Similarly, if the engineering group becomes subordinate to the research and development group, you can reflect this change by creating an empty directory named /**.:/rnd/eng** and then merging the contents of the /**.:/eng** directory into /**.:/rnd/eng**, effectively appending /**.:/eng** below /**.:/rnd**.

For additional information, see:

- “Overview of the Merge and Append Procedures” on page 5-55
- “Merging CDS Directories” on page 5-46
- “Error Handling” on page 5-33
- “Merging CDS Directories into a Foreign Cell” on page 5-49
- “Restoring Merged CDS Directories” on page 5-60

Updating CDS Data

When names exist in the namespace, users who have the appropriate access can make changes to the data that is associated with the names. Any addition, modification, or deletion of CDS data initially happens in only one replica — the master replica. The following sections introduce the main methods by which CDS keeps other replicas consistent — update propagation and the skulk operation. They also describes two timestamps that help to ensure consistency in CDS data.

- “Update Propagation” on page 5-69
- “Skulk Operation” on page 5-63
- “Keeping Data Consistent Through Timestamps” on page 5-37

By understanding the concepts in these sections, you can more effectively plan the content and replication of directories in your namespace.

Distributed Time Service (DTS) Overview

DTS is a software-based service that provides precise, fault-tolerant clock synchronization for systems in Local Area Networks (LANs) and Wide Area Networks (WANs). The clock synchronization that is provided by DTS enables distributed computing applications to determine event sequencing, duration, and scheduling.

DTS consists of software components on a group of cooperating systems. It conforms to the client/server model used in the Distributed Computing Environment (DCE). In the DTS implementation, each server supplies the time to many client systems and applications through intermediaries called *clerks*. Clerks reside on their client systems.

Note: Throughout the sections on DTS, the term *entity* is refers to the server or clerk process when they have the same functions.

Most DCE nodes have a DTS clerk that adjusts the clock on its client system. Clerks use Remote Procedure Calls (RPCs) to obtain time values from one or several servers in the network. The nodes that do not have DTS clerks have DTS servers. In addition to providing time values to clerks, servers also adjust the system clocks on their host systems. Servers are also able to obtain reference time values from sources of standardized time that are outside of the network.

Because no device can measure the exact time at a particular instant, DTS expresses the time as an interval that contains the correct time. In the DTS model, clerks obtain time intervals from several servers and compute the intersection where the intervals overlap. Clerks then adjust the system clocks of their client systems to the midpoint of the computed intersection. When clerks receive a time interval that does not intersect with the majority, the clerks declare the nonintersecting value to be faulty. Clerks ignore faulty values when computing new times, thereby ensuring that defective server clocks do not affect clients.

DTS also permits the importation of time values from outside sources, such as the U.S. National Institute for Standards and Technology (NIST). DTS uses the Coordinated Universal Time (UTC) standard that has largely replaced Greenwich Mean Time (GMT) as a reference. Many standards' bodies disseminate UTC by radio, telephone, and satellite; commercial devices (time providers) are available to receive and interpret these signals. DTS offers a Time-Provider Interface (TPI) that describes how a time-provider process can pass UTC time values to a DTS server and propagate them in the network. The TPI also permits other distributed time services to interoperate with DTS.

DTS provides many valuable services for computer networks that run distributed applications. The major features and benefits of DTS include:

- **Correctness** — DTS maximizes the probability that a client will receive the correct time. DTS uses Coordinated Universal Time (UTC) as a base reference and defines any time interval containing UTC as correct.
- **Quantitative Time Measurement** — DTS uses specific measurement and manufacturer's specifications to determine the quality of the times that are reported by servers.
- **Fault Tolerance** — DTS reports faulty servers and does not use their time values during clock synchronizations.
- **Management Capability** — The DCE and DTS control programs (*dcecp* and *dtscp*) enable you to control and monitor the software.
- **Application Programming Interface (API)** — DTS provides an interface that allows applications to obtain, compare, calculate, and display UTC time values.
- **Local Time Translation** — When displaying time values, DTS translates the UTC times it uses internally into local time values.
- **Monotonicity** — DTS normally provides unidirectional clock adjustment. You can use the DCE or DTS control program, though, for nonmonotonic clock adjustment.
- **Automatic Configuration** — DTS entities use RPC profiles (search tables) to obtain the locations of servers in a local area or cell.
- **Efficiency** — Complexity is placed in the servers; network overhead is minimal.

For additional information, see:

- “DTS Advantages” on page 6-4
- “DTS Operations” on page 6-9

DTS Usage

The following describes system clock and network characteristics, DTS synchronization concepts, DTS clock adjustment, and DTS time representations. System managers need to read this section to gain a basic understanding of DTS concepts before reading “Managing the DCE Distributed Time Service” on page 1-43.

Time Measurement Factors

The following describes the factors that affect time measurement and explains how DTS handles them.

Clock Error: All system clocks have common properties that contribute to clock error and interfere with the synchronization process. System clock error seems to increase over time. The rate of change of error is known as *drift*. If each system clock in a network started at the same time and ran at the same rate, the clocks would remain synchronized. Because each system clock drifts at a different rate, however, the system clocks throughout a network become desynchronized.

The difference between any two clock readings is known as the *skew* between the clocks. Typically, the clocks that are used in many computer systems have a manufacturer-specified maximum drift of a few seconds per day. If uncorrected for several days, the skew between networked system clocks can inhibit the performance of distributed applications.

The DTS server or clerk on each node tracks the drift of its client's system clock and periodically synchronizes with other DTS nodes to reduce the skew between its client's time value and those of the other DTS nodes. The DTS entity adjusts the system clock on its client node as the final step in this repeating synchronization process.

Communications and Processing Uncertainties: Communications delays also inhibit the synchronization process, especially when two systems communicate over a WAN or low-speed link. DTS can adjust for the known processing delays that are required to send and receive messages between systems. Because of the varying quality of communications links, however, the time that is required to send, receive, and acknowledge messages varies from one message to the next. These delays cannot be known exactly because transit over the network and the time required to read an incoming timestamp both vary.

Rather than using estimates of communications and processing delays, DTS records all known error factors that accompany a time measurement sent over the network. This measurement enables DTS to determine the relative quality of a time source regardless of its geographic location or changing conditions on the communications links.

Inaccuracy Values

In order to synchronize system clocks to the most accurate settings, DTS needs a way to determine the accuracy of time sources relative to each other and to UTC. This section describes how DTS determines the relative accuracy of any time source that is available in the network.

DTS uses an inaccuracy value (or tolerance) to determine the relative precision of time values that it obtains from system clocks and external time providers. This DTS feature effectively transforms each time value into an interval (or range) rather than a point on a continuum.

Inaccuracy values are determined by the following three factors:

- **Drift** — When reading a clock, DTS calculates the maximum amount of time that the clock may have drifted since DTS previously read the clock. Drift is the largest component of most inaccuracy values.
- **Communications Delay** — The inaccuracy also contains the uncertain portions of the communications delays between systems. Although DTS compensates for processing delays, it cannot predict or directly measure the varying delays that occur on network links. The inaccuracy values that a clerk or server obtains from systems on a LAN tend to be much lower than those obtained from servers outside the LAN.
- **Leap Seconds** — UTC time is measured by atomic clocks, which are extremely stable. The standard, however, keeps time based on the earth's position. Due to the slowing of the earth's rotation, it occasionally becomes necessary to advance UTC by 1 second. These events are known as *leap seconds*. Leap seconds may occur in the final second of any month and normally occur about once every 18 months. At the end of each month, DTS accounts for leap seconds by increasing all inaccuracy measurements by 1 second. DTS later adjusts the clocks to remove the extra second of inaccuracy if an external time provider determines that a leap second did not actually occur.

Without DTS to correct it, a system clock's inaccuracy is always increasing. For example, suppose that a clock starts with a UTC time of 0:00:00.00 (midnight) and zero inaccuracy. Because of drift, when the clock next shows a time of 0:00:00.00, the inaccuracy is 8 seconds. UTC time may be 23:59:52.00 or 0:00:08.00, but it is probably somewhere in between. Therefore, the system time is an interval that contains UTC time and is bounded by the inaccuracy, as shown in Figure 1-4. Using the DTS format for displaying time, the combined time and inaccuracy interval is expressed as:
1993-08-03-00:00:00.000I08.000.

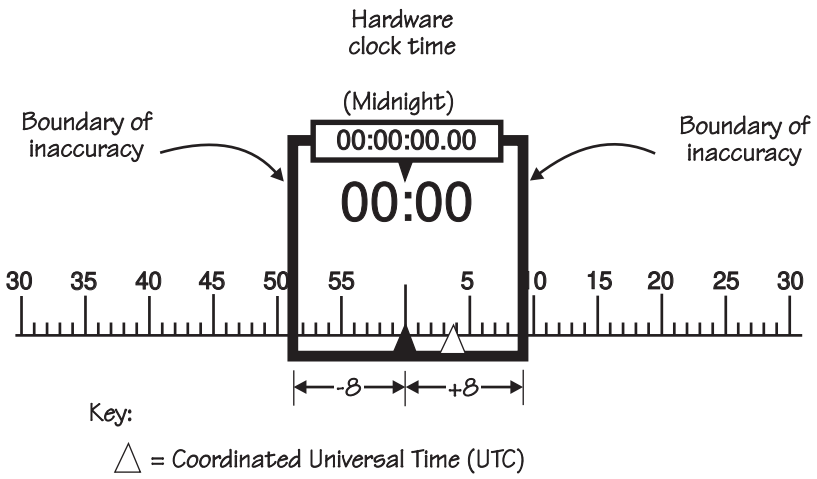


Figure 1-4. Time and Inaccuracy

Synchronizing System Clocks

To maintain uniform system times, DTS servers and clerks periodically synchronize the clocks in all network systems. The DTS entity that is on each system performs these synchronizations by requesting that servers send their combined clock and inaccuracy values (time intervals) to the originating system. The entity then uses the values that are sent by the servers to compute a new system time.

DTS servers and clerks have slightly different synchronization procedures. Before attempting to synchronize with other systems, DTS servers always check to see if an external time provider is present on the server system. A given server requests times from other servers if no time provider is available.

When no time provider is available and a server synchronizes with its peer servers, the server uses its own system time as one of the input values when it computes a new system time.

Most network systems run the DTS clerk process. Clerks cannot have time providers and they do not use the system time of their client systems to compute new times. When a clerk is synchronizing its client system's clock, the clerk uses only the time values that it obtains from servers to compute a new system time.

When a DTS clerk requests time intervals from several servers, it uses them to calculate a new time that is correct (that is, contains UTC) and that minimizes inaccuracy. When the servers respond and the DTS entity calculates network communications uncertainties and drift for each of the time values, the clerk has a set of intervals (**t1** through **t4** in Figure 1-5). Since each interval contains UTC, the intersection is the smallest interval the clerk can choose that also contains UTC. This intersection is the computed time. The DTS entity uses the computed time interval to adjust the clock on the system that receives the server values.

In addition to eliminating large inaccuracy values during synchronization, DTS also discards intervals that are received from faulty clocks (**t2** in Figure 1-5). DTS detects and rejects clock intervals that do not intersect with the majority of the intervals. When DTS detects a faulty interval, it notifies the system manager by displaying an event message, identifying the server that sent the faulty value.

A server that has a high-drift clock or is far away in the network submits its time to the DTS entity (**t1** in Figure 1-5), but the large time interval is ignored because more accurate times are available. Note that, in Figure 1-5, endpoints of correct time (**t1**) are further from the computed time midpoint than those of the interval that is declared faulty (**t2**).

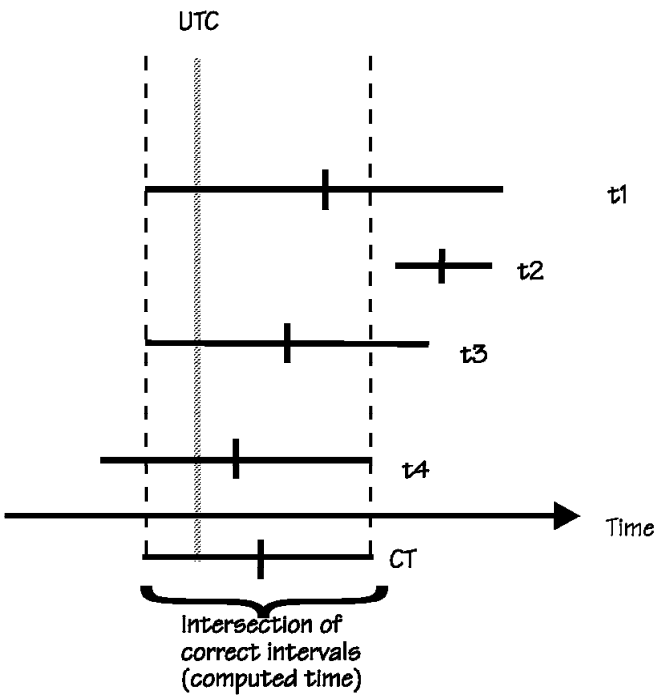


Figure 1-5. Computed Time

During the synchronization process, servers with the greatest accuracy have the most influence in determining new system times throughout the network.

In Figure 1-5, the server that submitted time value **t3** has the smallest correct interval and is, therefore, the closest to the computed time. Server systems with external time providers are usually the servers with

the most accurate times. Beyond TP servers, those servers with the highest quality clocks and best communications links tend to influence the time on other systems to the greatest degree.

The synchronization process also reduces the skew between systems. The computed time interval is often smaller than the interval that is supplied by any single clock. Note that the computed time in Figure 1-5 on page 1-38 is a smaller interval than any of the source intervals. As the synchronization procedure is repeated on each network system, the skew between systems is reduced and they are more closely synchronized. However, if a time provider is absent from the network, the clocks may collectively drift away from UTC.

How DTS Adjusts System Clocks

Many system clocks are based on an oscillator and operate with a combination of hardware and software. The hardware for each clock contains a timer that sends interrupts to the operating system at fixed intervals; each interrupt is a single tick. A software register that contains the current value of the time is incremented by a fixed amount (for example, 10 milliseconds) at each tick. DTS adjusts the rate of the clock by changing only the incremental value that is added to the software register. DTS does not directly affect the ticks of the hardware clock.

DTS adjusts system clocks at the rate of 100 to 1 (that is, it requires 100 time units to adjust 1 time unit of error). For example, it takes 1 minute and 40 seconds to correct a 1-second error. This rate of adjustment exceeds the normal rate of drift so that synchronization is carried out without further significant interference from the clock.

Figure 1-6 illustrates how DTS changes the increment to the software register. The top line represents a 10-millisecond increment to the normal clock at every 10-millisecond tick. The middle line illustrates the adjustment to a fast clock. DTS slows the clock by incrementing the register by 9.9 milliseconds instead of 10 milliseconds at each tick. The bottom line illustrates the adjustment to a slow clock. DTS speeds it up by incrementing the register by 10.1 milliseconds instead of the usual 10 at each tick.

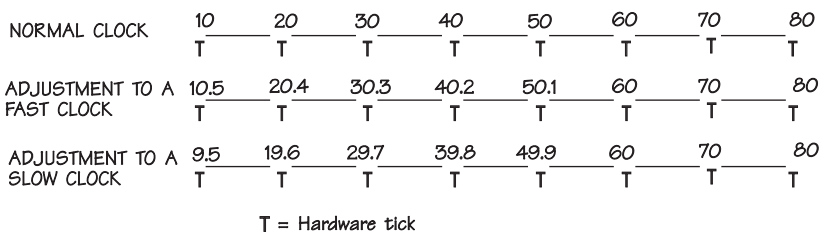


Figure 1-6. Adjustment of the Clock

Note: DCE for AIX provides only for smooth adjustment of the clock as seen by DCE components and by applications using the DTS API. The actual system clock is updated abruptly.

It is occasionally preferable to set the system clock immediately, rather than adjusting it gradually. DTS provides this option for the following situations:

- During system startup when you want to set the initial system time;
- If it has been a long time since the last synchronization and you decide that the skews between system clocks are too large to wait for a gradual adjustment;
- When a network has had catastrophic hardware problems, causing a large number of the clocks to become faulty; and

- When the time interval for a given clock does not intersect with the intervals of other clocks and the error exceeds a predetermined tolerance.

DTS Time Representation

Coordinated Universal Time (UTC) is the international time standard that has largely replaced Greenwich Mean Time (GMT). The standard is administrated by the International Time Bureau (BIH) and is in widespread use. For all its internal processes, DTS uses opaque binary timestamps that represent UTC. You cannot read or disassemble a DTS binary timestamp. The DTS API allows other applications to convert or manipulate the timestamps, but they cannot be displayed. To display the binary timestamps, DTS translates them to ASCII text.

Absolute Time: An absolute time is a point on a time scale. For DTS, absolute times reference the UTC time scale; absolute time measurements are derived from system clocks or external time providers. When DTS reads a system clock time, the time is recorded in an opaque binary timestamp that also includes the inaccuracy and other information. When you use the **dcecp clock show** command to display an absolute time, it is converted to ASCII text, as shown in the following example:

```
1993-11-21-13:30:25.78523-04:00I010.0825
```

DTS displays all times in an ISO-compliant format. The International Organization for Standardization (ISO) format that generated the previous display example is detailed in Figure 1-7.

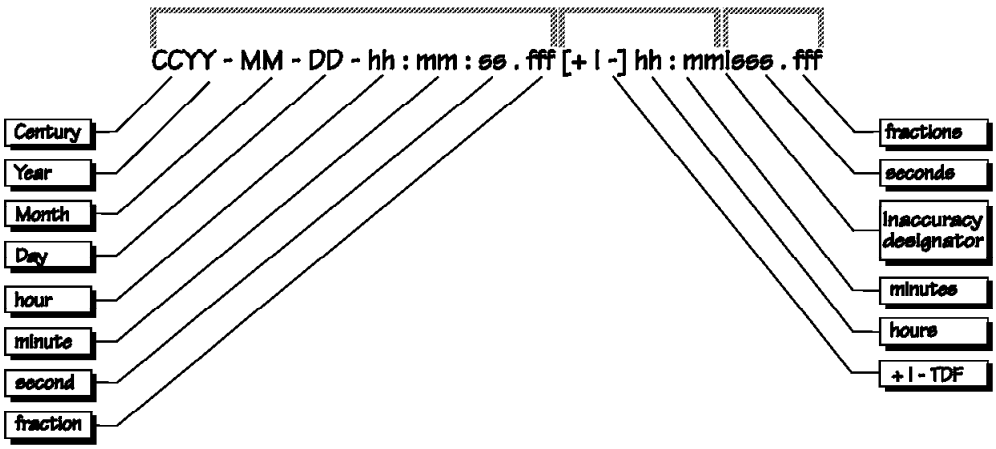


Figure 1-7. ISO-Compliant Time Format

In the format example, shown in the preceding figure, the relative time preceded by the plus sign (+) or the minus sign (-) indicates the hours and minutes that the calendar date and inaccuracy are offset from UTC. The presence of one of these characters in the string also indicates that the calendar date and time are the local time of the system, not UTC. The delineator (I) indicates the beginning of the inaccuracy component that is associated with the time.

You can express the DTS time that you want to display in several ways. The DTS time BNF format is defined in "DTS Extended BNF" on page A-2.

Although the **dcecp** program's **clock show** command displays all times in the previous format (see Figure 1-7), the interface also accepts the following variations to the ISO format on input, as shown in Figure 1-8 on page 1-41.

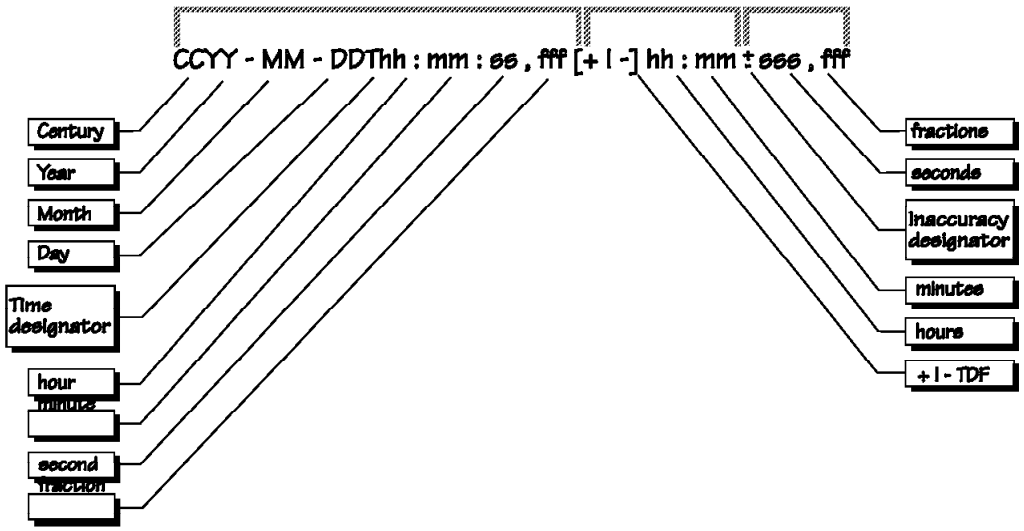


Figure 1-8. ISO-Compliant Time Format Variation

In the preceding example, the delineator **T** separates the calendar date from the time, the comma (,) separates seconds from fractional seconds, and the plus-or-minus sign (±) indicates the beginning of the inaccuracy component.

DTS offers a translation feature that changes UTC-based absolute times to your local time whenever the time is displayed. The local time displayed is derived from UTC plus a Time Differential Factor (TDF), which can have a positive or negative value. In the previous example, the string ([+|-] hh:mm) denotes the TDF. When installing a system, you select a time-zone rule for the system, which determines the TDF and any seasonal changes to the TDF. After the initial startup, all subsequent output times reflect the local time. If an absolute time is displayed by your system, and it does not contain TDF information, it is a UTC time.

Relative Time: A relative time is a discrete time interval that is usually added to or subtracted from another time. The TDF that is associated with absolute times is an example of a relative time. Relative times are normally used as input for commands or system routines. Figure 1-9 shows the format for relative time.

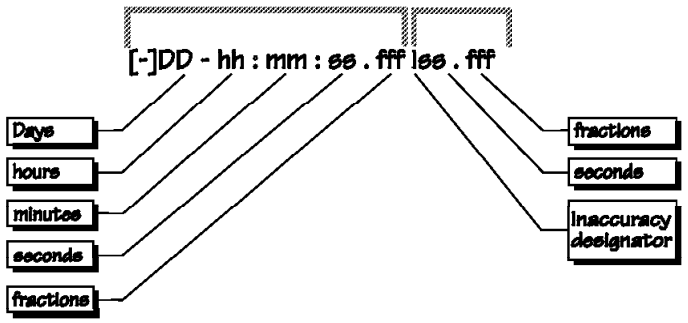


Figure 1-9. Relative Time Format

The simple relative times that you specify with DTS-related **dcecp** program commands do not use the calendar date nor inaccuracy fields because these fields are associated with absolute times. Positive relative times are not signed, but negative relative times are preceded with a minus sign (-).

IBM DCE for AIX adjtime() Limitations

The Distributed Time Service uses the IBM DCE for AIX system call **adjtime()** to make adjustments to the system clock. If a program samples the system clock more often than once every 10 milliseconds while **adjtime()** is performing a negative adjustment, the program may observe negative discontinuities in the system time.

DTS is designed on the premise that **adjtime()** updates the system clock at a 100-to-1 ratio. However, when **adjtime()** receives a request to update the system clock, the system call may alter the requested rate, depending upon the size of the update. Because DTS cannot determine when **adjtime()** alters the update rate, DTS may report that it is in the updating state for a period of time after the update has actually completed. This characteristic of **adjtime()** does not diminish the ability of DTS to update the system clock correctly.

Example

The following example shows a relative time used in a typical DTS-related **dcecp** command:

```
21-08:30:25.000
```

Simple relative times are often subtracted from or added to other relative or absolute times. For example, if you say "I will meet you in an hour," you add a relative time of +01:00 to the present absolute time. In the case where you add or subtract a relative time and an absolute time, note that the inaccuracy of the input absolute time is carried over to the resulting absolute time (for example, 1993-11-30-00:30:25.000I00.030 minus 00-00:15:25.000 equals 1993-11-30-00:15:00.000I00.030).

Interoperating with Network Time Protocol

Network Time Protocol (NTP) is an Internet-recommended standard. The NTP synchronization subnetwork is represented by a tree-structured graph with nodes representing time servers and edges representing the transmission paths between them. The root nodes of the tree are designated primary servers that synchronize to a radio broadcast or calibrated atomic clock. Remaining nodes are designated secondary servers that synchronize to other servers (primary and secondary).

The number of subnetwork hops between a particular server and a primary server determines the stratum of that server (that is, the smaller the number of hops, the lower the stratum). A lower-stratum server always has a higher accuracy than a higher-stratum server. All servers have identical functionality and can operate simultaneously as clients of the next lower stratum and servers for the next higher stratum.

Servers, both primary and secondary, typically run NTP with several other servers at the same or lower stratum. A selection algorithm attempts to select the most accurate and reliable server (or set of servers) from which to actually synchronize the local clock.

You can use both NTP and DTS in large computer networks that have embedded local nets (that is, those connected by routers, gateways, and bridges) and use both broadcast and point-to-point transmission media. DTS and NTP can run simultaneously on the same LAN. The following sections describe how to give time to and get time from local and remote NTP time sources and how to prevent loops.

- "Getting the Time from NTP Time Sources" on page 6-13
- "Giving the Time to NTP Nodes" on page 6-15
- "Preventing Loops" on page 6-28

Managing the DCE Distributed Time Service

The DCE control program (**dcecp**) has commands that you can use to perform these tasks. This guide contains brief descriptions of these commands. Detailed descriptions of the command appear in the *DCE for AIX Administration Command Reference*.

Prior to the **dcecp** program's creation, the DTS control program (**dtscp**) was used to manage DTS. You can still use this control program, but all of its operations have been incorporated into the **dcecp** program. Again, you can refer to the *DCE for AIX Administration Command Reference* for detailed descriptions of the **dtscp** program's commands for managing DTS.

For additional information, see:

- “Managing DTS with **dcecp** Commands” on page 6-17
- “DTS Timestamp Format” on page 6-12
- “Reconfiguring DTS on Nodes” on page 6-30
- “Temporarily Reconfiguring DTS” on page 6-31
- “Modifying Clerk and Server Attributes” on page 6-21
- “Management Tasks Specific to Servers” on page 6-19
- “Changing the System Time” on page 6-1
- “Controlling Access to DTS” on page 6-4

Planning Your DTS Implementation

Two main categories of personnel interact with the DTS software — system managers and applications programmers. Programmers usually do not need to be involved in the planning stages of the DTS implementation. If you are writing a program to import a source of UTC time into the service, however, you may want to locate the time provider at the server that is closest to the programmer. Close proximity to the time provider helps the programmer when testing the software application with the time provider hardware.

System managers or network architects usually plan the DTS implementation. They decide which nodes are servers and which are clerks, and decide how the DTS implementation grows with the network. DTS is scalable for large networks so that expanding an implementation to include new nodes is relatively simple.

System managers also install the software and maintain DTS. As the network grows, system managers ensure that the service is running with acceptable accuracy and install new servers, time providers, and clerks.

The following sections explain how to plan your DCE Distributed Time Service (DTS) implementation, including personnel selection for the planning process and planning for DTS on a LAN, an extended LAN, or a WAN.

- “General DTS Planning Guidelines” on page 6-12
- “Configuring DTS for a LAN” on page 6-3
- “Configuring DTS for an Extended LAN” on page 6-3
- “DTS Configuration for WANs and WAN Links” on page 6-6
- “DTS Planning for External Time Providers” on page 6-11

Since DTS installation is described in the *DSS for AIX Up and Running!*, installation considerations are only included in this guide by reference. It is important to note, however, that many of the planning considerations for DTS are tied to the overall planning of the DCE, especially the CDS and Security components. See “DCE for Administrators Overview” on page 1-1 for background information on the interdependencies between the DCE components.

DCE Security Overview

DCE Security consists of the following services:

- The Registry Service maintains the registry database, which is a replicated database of principals, groups, organizations, accounts, and administrative policies.
- The Authentication Service handles user authentication (the process of verifying that users are correctly identified). The Authentication Service also issues tickets that a principal uses to access remote services. The ticket contains data that is presented by the principal requesting the service to the principal providing service.
- The Privilege Service supplies the user's privilege attributes which ensure that a principal has the rights to perform requested operations.

In addition, DCE Security provides:

- The Access Control List (ACL) Facility which establishes and grants access rights to an object based on the object's access permissions, and
- The Extended Registry Attribute (ERA) Facility which provides tools to extend the registry database schema to define additional attributes and tools to attach those attributes to registry objects.

The DCE host daemon (**dced**) acts as the Security client. The Registry, Authentication, and Privilege Services are implemented as a single daemon — the Security server (**secd**).

For additional information, see:

- "The Registry Database" on page 7-142
- "Physical Security of the Database" on page 7-103
- "How the Registry Database is Stored" on page 7-88
- "Replicated Databases" on page 7-126
- "The /**etc**/**passwd** and /**etc**/**group** Files and the Registry" on page 7-141
- "The Local Registry" on page 7-141
- "Security Object Names" on page 7-135

Accessing Registry Objects

The following sections describe the permissions that apply to objects in the registry. Because the permissions that are granted are based on the way the registry database is structured, these sections briefly describe the structure of the registry database. They also describe the permissions for each object in the registry database, the registry ACL Managers, and the initial registry ACLs.

- "The Registry Database" on page 7-142
- "Registry Permissions" on page 7-114
- "Registry ACL Manager" on page 7-112
- "Initial Registry ACLs" on page 7-89

Both the **dcecp** command and the **acl_edit** command have functions for creating, modifying, and deleting ACL entries for registry objects. See the page in the *DCE for AIX Administration Command Reference* for each command for descriptions of the operations they perform on ACL entries.

Administering a Multicell Environment

Other sections in this guide describe the DCE administration tasks that are performed within individual cells. The administration of a multicell environment (one in which principals from foreign cells access objects in the local cell) has additional tasks and considerations that arise from the interaction of principals across different cells.

In fact, you can have two types of system administrators — one for local cell administration and one for intercell administration of the multicell environment. If you set up groups for the two types of administrators, you can set the ACL for the **krbtgt** directory (which contains cell principals) in the registry database to allow updating by only the group of intercell administrators. Be sure, however, to allow all other users read access to the **krbtgt** directory or intercell access will be denied to those users. If you protect the **krbtgt** directory in this way, ensure that all directories below the **krbtgt** directory also have the proper ACLs. The easiest way to accomplish this is to change the object ACL and the initial creation ACLs on **krbtgt** directory after the registry is created.

The following sections describe the trust relationship between cells that allow principals from foreign cells access to objects in your cell and vice versa.

- "Trust Relationships" on page 7-143
- "Establishing Trust Relationships" on page 7-75
- "Creating Trust Relationships" on page 7-60
- "Modifying Cross-Cell Authentication Accounts" on page 7-94

Authorization Overview

An ACL contains a list of entries that specify the principals who can access an object and the operations that those principals can perform. The principals can be named explicitly or be members of a group that is identified in the ACL entry. The ACL is associated with the object it protects. The operations a principal can perform are specified by permissions.

DCE permissions can be set for the following:

- Owner, group, and other
- Specific individual principals in the local cell and in foreign cells
- Specific individual groups in the local cell and in foreign cells
- Any other principals in a specific foreign cell for whom individual permissions have not been set
- Any principals in any cell who have been authenticated by the DCE Authentication Service
- Delegate users, servers, or groups, in local or foreign cells
- Unauthorized users

ACLs also provide a masking capability and a method for integrating protections from DCE versions that are different from the current version.

File systems are frequently designed to provide access permissions for file system objects, such as files and directories. ACLs in DCE are more extensive. In DCE, many objects can have ACLs and can be assigned permissions. DCE ACLs control access to objects managed by DCE components, like the Distributed File Service, the Security Service, and the Directory Service. ACLs for the Security Service (the component that controls accounts) can, for example, authorize certain principals to change all of the information associated with an account, authorize other principals to change only a subset of the information associated with accounts, and restrict other principals from changing any of the information associated with accounts.

DCE can support particular sets of permissions that correspond to particular types of objects. For example, for containers there can be an insert permission that other objects, such as principals, do not need. This extensive usage of ACLs is in contrast to that of POSIX systems, for example, where only file system objects are protected by permission bits, with a standard set of permissions (read, write, and execute) being used. The DCE control program has a command, **acl permissions**, that shows the permissions specific to the ACL associated with the named object.

ACL Managers

An ACL Manager is that portion of a server that handles ACLs. One ACL Manager can support several different types of ACLs. From a more abstract point of view, each ACL type is supported by a corresponding ACL Manager Type. Informally, ACL Manager Types are sometimes called ACL Managers. Figure 1-10 shows ACL Managers in servers.

The client side allows you to connect to any server exporting the ACL interface so that one program can manipulate all ACLs. The **dcecp** and **acl_edit** programs use the feature.

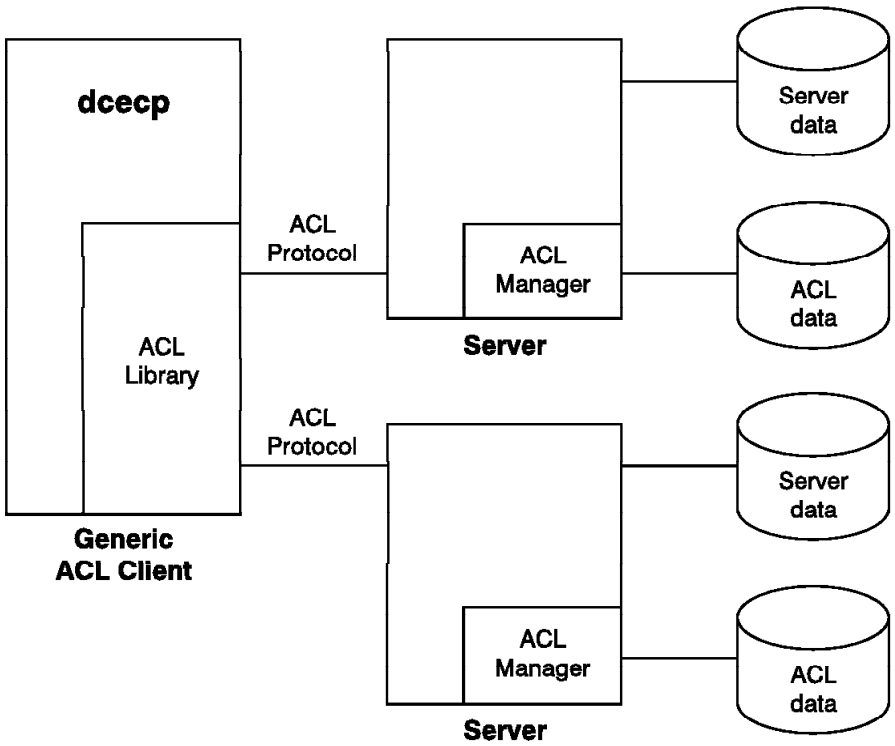


Figure 1-10. ACL Managers in Servers

In addition to the standard DCE components, ACLs can control access to any object for which an ACL Manager has been implemented. ACLs can be associated with user-written applications to protect access to the use of the application itself, the files in the application, and even fields in those files.

All of the elements of ACLs described in this section are available to ACL Managers; however, each Manager may implement all or only a subset of the elements. For information on how ACLs are used by specific DCE components, consult that component's section of this manual.

ACL Interpretation

Part of the information associated with an account is a principal and a set of groups. (The groups are called a *project list* in this context, in honor of its Multics origin.) Together, the principal and project list are called the *privilege attributes* (or client-side access control information) associated with the account. The principal and each of the groups is represented by both a string name and a UUID. The privilege attribute UUIDs are contained in the credentials that are used in authenticated Remote Procedure Calls (RPCs). Servers grant access based upon the contents of credentials received in RPCs. Although servers typically reject unauthenticated RPCs, any server can support a policy of accepting them. In that case, the server's ACL Manager must support the **unauthenticated** mask ACL entry type so that the server can further restrict the access granted to such unauthenticated clients.

When a principal requests access to a DCE object associated with an ACL, the object's ACL Manager compares the UUIDs of the principal and any groups of which the principal is a member (the principal's privilege attributes) with the UUIDs of the principals and groups listed in the ACL entry. It does this simply by reading through the list of ACL entries. The Manager grants the access permissions in the first ACL entry (or entries in the case of groups) it finds that match any of the principal's privilege attributes. If the permissions in the matching entry allow the requested mode of access, the principal gains access; if not, access is denied.

Credentials Inherited by Processes

Processes created or spawned by a principal inherit the principal's credentials. For example, if you log in, are authenticated, and start an application, the application you start inherits your authenticated credentials and runs as though it were you. The application's permissions for any given object are the same as your permissions. Processes spawned by the application carry your identity and pass it down to processes they start.

Note: Changing the **setuid** permission bit changes only the local operating system identity under which an executable file runs, not the network identity.

Some servers are written to run as separate authenticated principals. For these servers, the system administrator creates an account in the registry database. After you start these servers, the server process authenticates with the registry, receives its credentials, and runs under its own identity, not yours.

Control Programs for Managing the Security Service

You can perform most of the management tasks for the Security Service using the DCE control program (**dcecp**). However, some of the components of this service require you to use other control programs provided in DCE.

See the following sections for detailed information:

- “DCE Control Program Usage” on page 2-15
- “Registry Editor Usage” on page 7-113
- “**sec_admin** Program Usage” on page 7-136

Control programs that you use for Security-related management tasks from time-to-time, such as **password_export** and **sec_create_db**, are not covered in these sections. These programs are described in other sections of this guide along with the instructions for performing the tasks.

Creating and Maintaining Accounts

All principals have two identities — a network identity that provides the ability to access DCE objects on machines across the network, and a local identity that provides the ability to access objects on the local machine. The two identities exist independently of each other, but in tandem. A principal's network identity is defined by an account in the network registry. A principal's local identity is defined by local data, such as entries in the **/etc/passwd** and **/etc/group** files stored in the local machine. If the **passwd_export** command is used to update the **/etc/passwd** and **/etc/group** files with data that is stored in the local registry, local identity data is derived from information that is stored in the network registry.

Registry accounts define a network identity by associating a principal with a group, an organization, and related account information, such as the password that is used to authenticate a principal's identity. You must create a registry account for any principal that engages in communication across the network, regardless of whether the communication is authenticated. The principals for which you must create registry accounts are:

- Each human user who accesses objects across the network (this probably includes all human users unless you are specifically restricting a user to the local machine);
- Each server that accesses objects across the network and runs under its own identity, not the identity of the principal who started it;
- Each machine in the network; and
- Any cell with which you engage in authenticated cross-cell communications. (Accounts for cross-cell authentication are special types of accounts that are described in “Administering a Multicell Environment” on page 1-45.)

For additional information, see:

- “User Accounts” on page 7-146
- “Server Accounts” on page 7-137
- “Machine Accounts” on page 7-90
- “How Identities Represented by Accounts are Authenticated” on page 7-85
- “Adding Accounts” on page 7-9
- “Creating, Maintaining, and Deleting Keytab Files” on page 3-5
- “Maintaining the Local Registry” on page 7-92

Creating and Maintaining Principals, Groups, and Organizations

The following sections describe how to use **dcecp** to create and maintain principals, groups, and organizations.

- “Principal, Group, and Organization Names” on page 7-106
- “Reserved Principals and Accounts” on page 7-129
- “Object Creation Quotas” on page 7-95
- “Universal Unique Identifiers and UNIX IDs” on page 7-145
- “Adding and Maintaining Principals” on page 7-16
- “Extended Security Attributes for Principals” on page 7-79
- “Adding and Maintaining Groups and Organizations” on page 7-14
- “Maintaining Membership Lists” on page 7-90
- “Creating and Maintaining Aliases for Principals and Groups” on page 7-46

Creating and Using Extended Registry Attributes

The Registry stores specific information about objects, including principals, groups, organizations, and accounts. This information is generated when you use the **dcecp** command to create principals, groups, organizations, and accounts. In addition to the fixed attributes that are associated with these objects, you can also define additional attributes and use **dcecp** to associate them with these objects as well. These additional attribute types are specified in registry schema definitions.

Using the Extended Registry Attribute facility, you can add schema entries that define attribute types of your choosing. These attributes are called extended attributes because they extend the registry to contain attributes beyond the fixed attribute set. After defining extended attribute types, you can attach them to a security object with the **dcecp create** or **modify** command. The extended attribute types you create are used by custom applications that run in conjunction with the DCE and can be accessed by those applications for processing. For example, if you work with an MVS application that requires a user's MVS name, you can establish an MVS name extended attribute and store it in the registry. The MVS name can then be passed to the MVS application for appropriate processing.

If a principal has extended attributes, these attributes can be carried with the Extended Privilege Attribute Certificate (EPAC) obtained when the principal is authenticated.

In this manual, attribute type refers to the schema entry that defines an extended attribute type. Attribute instance refers to an attribute that is attached to a registry object and has a value.

The following sections describe how to create and maintain attribute types and attribute instances.

- "The **xattrschema** Object" on page 7-143
- "Creating and Maintaining Attribute Types" on page 7-49
- "Defining Attribute Trigger Servers" on page 7-62
- "Creating and Maintaining Attribute Instances" on page 7-47

DCE Audit Service

Audit, a key component of DCE, is provided by the DCE Audit Service. Audit plays a critical role in distributed systems. Adequate audit facilities are necessary for detecting and recording critical events in distributed applications.

For additional information, see "Administration and Programming in DCE Audit" on page 7-19.

DCE Audit Service Administrative Tasks

The following administrative tasks are performed for the DCE Audit Service:

- "Setting DCE Audit Environment Variables" on page 7-138
- "Starting the Audit Daemon" on page 7-140
- "Controlling Access to the Audit Daemon" on page 7-42
- "Defining Event Classes" on page 7-65
- "Creating and Maintaining Filters" on page 7-54
- "Enabling and Disabling the Audit Logging Service" on page 7-74
- "Modifying and Querying Audit Daemon Attributes" on page 7-94
- "Controlling and Displaying Audit Trails" on page 7-44

DCE Audit Service Components

The DCE Audit Service has two basic components:

- **Application Programming Interfaces (APIs)** — Provide the functions that are used to detect and record critical events when the application server services a client. The application programmer uses these functions at certain code points in the application server program to actuate the recording of audit events. Other APIs can be used to create tools that examine and analyze the audit event records.
- **Audit Daemon** — The Audit daemon provides the following services:
  - Maintains the filters and the central audit trail file.
  - Exports an RPC interface with which it can be controlled by the DCE control program (*dcecp*).

DCE Audit Service Features

The DCE Audit Service has the following features:

- An Audit daemon (**auditd**) performs the logging of audit records based on specified criteria.
- Application Programming Interfaces (APIs) can be used as part of application server programs to record audit events. These APIs can also be used to create tools that can analyze the audit records.
- An administrative command interface to the Audit daemon directs the daemon in selecting the events that are going to be recorded based on certain criteria. This interface is accessed through the DCE control program (**dcecp**).
- An event classification mechanism allows the logical grouping of a set of events for ease of administration.
- Audit records can be directed to logs or to the console.

DCE Audit Service Overview

This section describes some of the concepts that are relevant to the administration of the DCE Audit Service.

Audit Clients

All RPC-based servers are potential audit clients--DCE servers and user-written application servers. DCE Security Service and the Distributed Time Service are auditable. That is, code points (discussed in the next section), are already in place in these services.

The Audit daemon can also audit itself.

Audit clients should have the **log** permission to the Audit daemon object to be able to use the central audit trail file. Permissions to the Audit daemon are discussed in the next section of this book.

Code Point

A code point is a location in the application server program where DCE Audit APIs are used. Code points generally correspond to operations or functions offered by the application server that requires audit. For example, if a bank server offers the cash withdrawal function **acct_withdraw()**, this function may be deemed to be an auditable event and be designated as a code point.

Code points are already in place in the DCE Security Service, Distributed Time Service, and Audit Service code. Code points and their associated events for the DCE Security Service are documented in the **sec_audit_events(5sec)** reference page of the *DCE for AIX Administration Command Reference*. Code points and their associated events for the DCE Distributed Time Service are documented in the **dts_audit_events(5sec)** reference page of the *DCE for AIX Administration Command Reference*. Code points and their associated events for the DCE Audit Service are documented in the **aud_audit_events(5sec)** reference page of the *DCE for AIX Administration Command Reference*.

Audit Event

An audit event is any event that an audit client chooses to record. Generally, audit events involve the integrity of the system. For example, when a client withdraws cash from his bank account, this can be an audit event because it can involve a possible security violation on the bank account.

An audit event is associated with a code point in the application server code.

Event Number

Every audit event is assigned an event number by the application programmer. The event number is a 32-bit integer, such as 0xC0000000. Event numbers are discussed in more detail in the *DCE for AIX Application Development Guide — Core Components*.

Event Class

Audit events can be logically grouped together into an event class. Event classes provide an efficient mechanism by which sets of events can be specified by a single value. Generally, an event class consists of audit events with some commonality. For example, in a bank server program, the cash transactions (deposit, withdrawal, and transfer) may be grouped into an event class. Event classes are discussed in more detail later in this book.

Event Class Files

Event classes are defined in event class files. All event class files must be created in the **/opt/dcelocal/etc/audit/ec** directory.

Default event class files are provided to classify auditable events from the DCE Security Service, Time Service, and Audit Service. They are installed on the host system when any of these services is installed.

The name of an event class is the same as its file name. Each event class is defined within an event class file.

You can define new event classes by removing or adding event numbers in the event class files, or by creating new event class files.

Event Class Names

Each event class has a symbolic name assigned to it. Following is the suggested name format of event classes that vendors should follow:

ec_*org_product_class*

where:

org Is the name of the organization or company that defines the event class.

product Is the name of the product for which the event class is defined.

class Is the characterization of the event class.

The following are two examples of event class names:

ec_osf_dce_authentication
Defines an authentication event class for OSF's DCE core components.

ec_transarc_encina_update
Defines an update event class for Transarc's Encina.

You may also define event classes to meet your own auditing needs. The following is the suggested name format for these event classes:

dce_*server-name_class*

where *class* is a characterization of the event class.

Event Class Numbers

If you define your own event classes, you must associate it with an event class number. Event class numbers are 32-bit integers. Each event class number is a tuple made up of a set ID and the class ID. The set ID corresponds to a set of event classes and is assigned by OSF to an organization or vendor. The class ID identifies an event class within the set of event classes. The organization or vendor manages the issuance of the class ID numbers to generate an event class number.

The structure and administration of event class numbers can be likened to the structure and administration of IP addresses. Recall that an IP address is a tuple of a network ID (analogous to the set ID) and a host ID (analogous to the class ID).

Event Class Number Formats

Event class numbers follow one of five formats (A to E), depending on the number of event classes in the organization. The format of an event class number can be determined from its four high-order bits.

Format A can be used by large organizations (such as OSF or major DCE vendors) that need more than 16 bits for the class ID. This format allocates 7 bits to the set ID and 24 bits to the class ID. Format A event class numbers with zero (0) as its set ID are assigned to OSF. That is, all event class numbers used by OSF have a zero in the most significant byte.

Format B can be used by intermediate-sized organizations that need 8 to 16 bits for the class ID.

Format C can be used by small organizations that need less than 8 bits for the class ID.

Format D is not administered by OSF and can be used freely within the cell. These event class numbers may not be unique across cells and should not be used by application servers that are installed in more than one cell.

Format E is reserved for future use.

The numbers with 110 in the most significant bits (that is, 0xC0000000 to 0xDFFFFFFF) are reserved to be used locally within a cell.

The event class number formats are illustrated in Figure 1-11 on page 1-53.

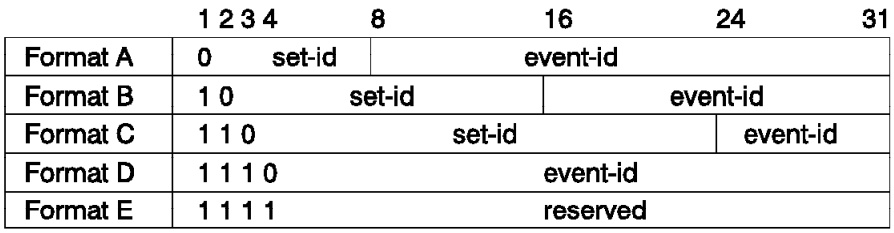


Figure 1-11. Event Class Number Formats

The cell administrator is responsible for administering and assigning local event class numbers and their names.

Filters

Once the code points are identified and placed in the application server, all audit events corresponding to the code points will be logged in the audit trail file, irrespective of the outcome of these audit events. However, recording all audit events under all conditions may neither be practical nor necessary. Filters provide a means by which audit records are logged only when certain conditions are satisfied. The administrator defines filters using the DCE control program.

A filter is composed of filter guides which specify these conditions. Filter guides also specify what action to take if the condition (outcome) is met.

A filter answers the following questions:

- Who will be audited?
- What events will be audited?
- What should be the outcome of these events before an audit record is written?
- Will the audit record be logged in the audit trail file, or displayed on the system console, or both?

For example, for the bank server program, you can impose the following conditions before an audit record is written:

> Log audit records on all withdrawal transactions (the audit events) that fail because of access denial (outcome of the event) that are performed by all customers in the DCE cell (who to audit).

Filter Subject Identity: A filter is associated with one filter subject, which denotes to whom the filter applies. The filter subject is the client of the distributed application who caused the event to happen. The filter subject has two parts — the filter type and the key.

There are eight filter types:

principal DCE principal in the local cell.

foreign_principal DCE principal in a foreign cell.

group DCE group in the local cell.

foreign_group DCE group in a foreign cell.

cell DCE cell in the network.

cell_overridable DCE cell in the network. This type can be overridden by a more specific filter type.

world All clients of the distributed application.

world_overridable All clients of the distributed application. This type can be overridden by a more specific filter type.

The key is the specific name of the **principal**, **foreign_principal**, **group**, **foreign_group**, **cell**, and **cell_overridable** filter types. The **world** and **world_overridable** filter types have no keys.

Filter Guides: A filter contains one or more guides. A filter guide contains three elements — audit condition, audit action, and event class.

An audit condition specifies the required outcome (or outcomes) of the event before an audit record is written to the audit trail. These outcomes are not mutually exclusive. The audit conditions are:

success Record only if event succeeds.

failure Record only if event fails.

denial Record only if event failed because of access denial.

An audit action specifies where the audit record is written. The audit actions are:

alarm Display the audit record on system console

log Log the audit record through an audit daemon or directly to an audit trail file.

The audit actions are not mutually exclusive and you can specify both of them.

The third element of the filter guide specifies the event class or event classes to which the filter will apply (for the specific filter subject identity).

Filter Rules: Filter rules are used to resolve overlapping guides from different filters. There are two filter rules — the override and the high-water-mark.

Under the override rule, filters that are overridable (that is, **cell_overridable** and **world_overridable** types) are nullified by more specific filters. The override rule serves as a mechanism that allows for complementary filters. A filter for a principal or a group is more specific than a filter for a cell or for the world.

The high-water-mark rule is applied after the override rule. If multiple filters are applicable to a client, the union of the actions (log or alarm) specified by these filters is applied.

A filter is applicable to a client if its principal, groups, or cell identity matches the key of the filter. The **world** and **world_overridable** filters have no keys and are applicable to all clients. If there are multiple filters that are applicable to a client, then the union of the actions (log or alarm) specified by these filters is taken.

Audit Trail File

The audit trail file contains all the audit records that are written by the Audit daemon. You can specify either a central audit trail file or a local audit trail file.

The central audit trail file is created by the Audit daemon when it is started. By default, if the **dce_aud_open()** function does not specify a name for an audit trail file, all audit records are sent to the Audit daemon which stores them in the central audit trail file.

If the **dce_aud_open()** function is invoked with a name for the trail file, this name becomes the path name to the local audit trail file and all audit records are sent to that file.

Examples

- The following is an example of a filter with two guides.

```
filter type: foreign_principal
key: /.../cell_x/foo
guide 1:
        audit conditions - denial
        audit actions - log
        event classes - Confidential
guide 2:
        audit conditions - denial
        audit actions - alarm, log
        event classes - Restricted
```

  Guide 1 specifies that an audit record is logged for any event in event class **Confidential**, if the user is the foreign principal **/.../cell_x/foo** and if the event fails because of access denial. Guide 2 specifies that an audit record is not only logged but is also displayed on the system console for any event in event class **Restricted**, for the same user and event outcome.

- The use of overridable filters is described in the following scenario:

  Alice in Company (cell) X is responsible for activating some operations (event class **critical_transactions**). Other principals in the company are also authorized to activate the same operations, but only under certain conditions, for example, when Alice is not available. The system administrator wants to log an audit record regardless of the event outcome (audit conditions = all) or who activates these operations. He also wants to generate an alarm if the activator is not Alice. This specification is implemented by the following two filters:

```
Filter 1:
         filter type: principal
         key: Alice
         guide 1:
                 audit conditions - all
                 audit actions - log
                 event classes - critical_transactions
Filter 2:
         filter type: cell_overridable
         key: X
         guide 1:
                 audit conditions - all
                 audit actions - log, alarm
                 event classes - critical_transactions
```

  When Alice invokes events in the critical_transaction event class, the principal filter (Filter 1) is applicable (because its key matches Alice's identity). The principal filter is more specific than the cell filter. Although the cell filter (Filter 2) is also applicable to Alice (Alice belongs to cell X), it is overridden by the principal filter because the cell filter is overridable. For other principals in Company (cell) X, the only applicable filter is the cell filter (Filter 2). Therefore, these same events cause an audit record to be logged and also raise an alarm.

  Nonoverridable world and cell filters are also useful. Without them, an administrator, for example, would have to delete all filters for groups and principals of a cell in order to make a cell-wide filter effective to the whole cell. (System administrators may want to introduce a temporary nonoverridable cell filter when a cell is suspected to be the source of a security problem.)

  The following figure illustrates the override relations between different types of filters. An arrow from filter type X to filter type Y means X overrides Y.

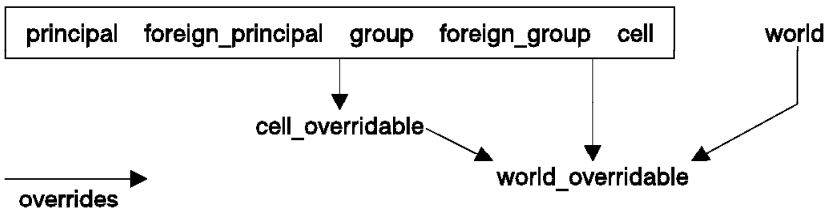


Figure 1-12. Override Relations Between Filter Types

DCE groups are generally defined for the purpose of granting access permissions. A group filter specifies *auditing the intent to use the group's privileges*, instead of specifying *auditing the principals that belong to the group*. That is, a group filter would not have auditing effects on a member principal of the group unless the principal has the intent to use the group's privileges (by including the group in the PAC). Because group filters are defined to audit the intention of using a group's privileges, they are independent of other filters and cannot be overridden.

DCE Authentication Service Servers and Clients

The Authentication Service consists of the registry database, Security servers, and Security clients. A security client communicates with a Security server (/**opt**/**dcelocal**/**bin**/**secd**) to request information and operations. Figure 1-13 is a simplified representation of the relationship between registry clients, servers, and a registry database. The Security servers access the registry database to perform queries and updates and to validate user logins. To gain access to the registry database, the Authentication Service must talk to the Registry Service. Figure 1-13 is a simplified representation of the relationship between Security clients, servers, and the registry database.

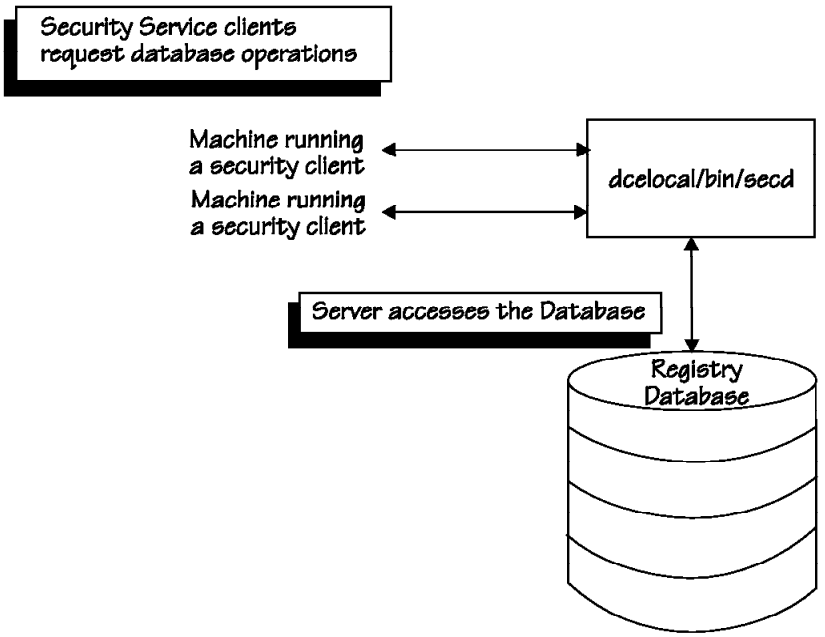


Figure 1-13. Machines, Servers, and the Database

Displaying Registry Information

Using the **dcecp** program, you can display information about the following Security objects:

- Principals
- Groups
- Organizations
- Accounts
- The registry
- The **xattrschema** object
- ACLs
- Keytab files

The following **dcecp** operations provide these displays:

- The **catalog** command displays the names of all the specified objects.
- The **list** command displays the names of the members of the specified groups or organizations or of the specified key table.
- The **show** command displays information about a specific instance of an object.

For additional information, see:

- "Displaying Account Information" on page 7-66
- "Displaying Group and Organization Information" on page 7-69
- "Displaying Principal Information" on page 7-71
- "Displaying **xattrschema** Information" on page 7-73
- "Displaying ACL Information" on page 7-68
- "Displaying **keytab** Information" on page 7-70

Handling Network Reconfigurations

The following sections describe the procedures to handle network reconfigurations that change the locations of registry replicas. Specifically, this includes:

- "Changing the Master Replica Site" on page 7-37
- "Removing a Server Machine from the Network" on page 7-125
- "Handling Network Address Changes" on page 7-84

To perform these procedures, you must be logged into the network registry account through an administrative account.

Importing AIX Accounts to DCE

The **passwd_import** command creates entries in the registry that are based on information in the **/etc/passwd** and **/etc/group** files. It provides a method of ensuring account consistency between machines that use the DCE Security Service and those that do not, as well as a means of adding an existing AIX user base to the registry.

For additional information, see:

- "**passwd_import** Operation" on page 7-100
- "The **passwd_import** Command Syntax" on page 7-141
- "**passwd_import** Usage" on page 7-102
- "Sample **passwd_import** Session" on page 7-131

Maintaining Policies and Properties

Registry policies are attributes that can be set registry wide. To provide a finer lever of control, policies can also be set for individual organizations and accounts. An organization's or account's policies can override the registry default policies if the organization's or account's policies are more restrictive.

Registry properties are attributes that apply to the principals, groups, and organizations created in the registry. They cannot be set for individual organizations or accounts. Properties regulate such things as the range of numbers that can be used for AIX IDs and whether encrypted passwords are displayed.

You can set both policies and properties with the **dcecp registry modify** command. In addition, you can set policies for an individual organization or account with the **dcecp organization modify** and **dcecp account modify** commands. In all commands, policies and properties to be set are supplied as attributes in standard **dcecp** attribute lists with the **–change** option or as attribute options.

For additional information, see:

- “Policies” on page 7-104
- “Authentication Policy” on page 7-33
- “Properties” on page 7-108

Performing Routine Maintenance

The following sections describe the security maintenance procedures that should be performed on a regular basis, such as:

- “Adding Accounts” on page 7-9
- “Overriding Entries in the Local /etc/passwd and /etc/group Files” on page 7-95
- “Changing the Registry Master Key” on page 7-38
- “Validating the Authenticity of the DCE Security Service” on page 7-146
- “Backing Up and Restoring the Registry Database” on page 7-36
- “Setting the **_s(sec)** Variable” on page 7-138
- “Ensuring Consistent Local Files” on page 7-74

Troubleshooting Procedures

The following sections contain procedures for troubleshooting the Security servers.

- “Restarting Security Servers” on page 7-130
- “Recovering the Registry” on page 7-112
- “Restarting the Primary Server in Locksmith Mode” on page 7-131
- “Automatic Changes to the Locksmith Account” on page 7-35
- “Starting a Security Server in Locksmith Mode” on page 7-140
- “Restarting a Security Server in Locksmith Mode” on page 7-130
- “Recovering the Master Replica” on page 7-111
- “Recovering Slave Replicas” on page 7-110
- “Turning a Master Replica into a Slave” on page 7-145
- “Forcibly Deleting a Slave Replica” on page 7-83
- “Restoring a Duplicate Master” on page 7-131
- “Adopting Registry Orphans” on page 7-22

Use these procedures only when network or hardware failures disrupt operations of the registry or when you encounter problems that can be remedied in no other way.

Before you run the procedures, ensure that you are logged in with an administrative account.

Using Access Control Lists

You can control access to DCE objects by using an authorization mechanism called an Access Control List (ACL). ACLs are associated with files, directories, CDS entries, and registry objects. They can be implemented also by arbitrary applications to control access to their internal data objects. Each ACL consists of multiple ACL entries that define who is authorized to do what to the object, specifically:

- Who can access the object,
- What kinds of access those principals or groups have to the object, and
- What kind of access is allowed to unauthenticated users.

For detailed information on how a specific DCE component implements the ACL authorization mechanism, see the component's part in this guide.

Note: In the discussions of DCE authorization in this guide, the term *user* is analogous to principal. A principal can be a human user, server, or a machine.

For additional information, see:

- "ACL Entries and Masks" on page 7-1
- "ACL Management Tasks" on page 7-9
- "Copying ACLs" on page 7-45
- "Container ACLs" on page 7-39

Reference

This section provides an introduction to reference information that affects various DCE components.

Object Identifier Files

The X/Open Directory Services (XDS) interface offers client application programmers the ability to create and maintain names in either CDS or GDS. Programmers also can create new CDS attribute names or GDS attribute type labels. In the DCE R1.1 Directory Service, every CDS attribute name and GDS attribute type label has a corresponding unique number called an *object identifier* (OID).

CDS provides a method for translating between object identifiers and human-readable names. This translation capability enables users to enter names instead of object identifiers at the DCE control program (**dcecp**) interface. Also, the **dcecp** program displays the names rather than object identifiers in command output. CDS attribute names and their corresponding identifiers are stored in a file called **cds_attributes**. GDS attribute type labels and their corresponding identifiers are stored in a file called **cds_globalnames**. See *DSS for AIX Up and Running!* for the full path names of all CDS files.

For additional information, see:

- "Origin of Object Identifiers" on page A-8
- "The **cds_attributes** File" on page A-9
- "The **cds_globalnames** File" on page A-11
- "Modifying the **cds_attributes** and **cds_globalnames** Files" on page A-8
- "Modifying CDS Entity Attributes" on page A-6

Time Providers and Time Services

The following sections explain the criteria to use when selecting a time provider and they describe time dissemination services, time providers (hardware and software), and their interaction with DTS.

- "Criteria for Selecting a Time Source" on page A-2
- "Sources of Coordinated Universal Time" on page A-8
- "World Time Zone Map" on page A-50

Chapter 2. The DCE Control Program

This section describes the control program tasks, utilities, and processes. For introductory information on the DCE control program, see “DCE Control Program Overview” on page 1-22. That section also groups related tasks together in the order in which they may normally occur so you can access these referenced tasks in a logical, tutorial-type manner. You may find this helpful in performing related tasks together from beginning to end.

Command Substitution Usage

Introduction — Command Substitution Usage

Command substitution provides a convenient way to express the return value of one command within another command. This is useful when you want to use the return value of one command as input to another command. Use brackets to invoke command substitution.

Another practical use of command substitution is to set up a test condition for an **if** statement. We show an example of this usage in “**if** Statements Conditionalize Your Scripts” under “More Detailed Information — Controlling Your Script” on page 2-2.

Examples — Command Substitution Usage

- The following example uses the **expr** command. Generally, **expr** performs a math function, returning the computed value expressed by its arguments.

```
dcecp> set a 4
4
dcecp> set b [expr $a+2]
6
dcecp> set b
6
dcecp>
```

- A more practical example might use command substitution for a command that returns a long name or a list. In the example in “**dcecp** Commands Usage” on page 2-19, the **[group list temps]** command returns a list to the **foreach** command which performs the **account modify** operation on each element in the list. We will look more closely at the **foreach** looping command later in this section.

```
dcecp> foreach i [group list users]
               {> account modify $i -change {expdate 1995-12-31}}
dcecp>
```

Controlling Your Script

Introduction — Controlling Your Script

The **dcecp** program provides several commands for controlling your script's execution. Commands such as **if**, **while**, **for**, **foreach**, and **case** executive parts of scripts under various conditions. The **break** and **continue** commands can stop execution of part or all of a command script.

More Detailed Information — Controlling Your Script

if Statements Conditionalize Your Scripts

Sometimes, you will want part of your script to execute only under certain conditions. Use an **if** statement to detect a condition and conditionally perform some operation. The syntax for an **if** statement is:

```
if test true_body else false_body
```

Loops Control Script Execution

Programming languages use loops to repeat operations as long as specified conditions exist. The **dcecp** program offers three kinds of loops: **foreach**, **while**, and **for**. The type of loop you use depends on the way conditions are specified.

The foreach *Loop*

When you want to perform a given operation on each element in a list, use the **foreach** command. Remember that a list is a collection of objects, or things typed in by you or returned from a command. The syntax is:

```
foreach variable_name list body>
```

The **foreach** command consists of a list, a script body, and a variable that represents each element of the list, in turn. The command runs the script body on the element represented by the variable and then sets the variable to be the next element in the list.

The while *Loop*

The **while** loop behaves like the while loop in C. It takes two arguments: an expression and a script (called the body). When the expression evaluates to nonzero, the while command executes the body and then reevaluates the expression, continuing the loop until the expression evaluates to 0. The syntax for a **while** loop is:

```
while expression body
```

The for Loop

The **for** loop also behaves just like its C counterpart. Although **for** is more complex than its sibling **while**, **for** keeps all of the loop control information together, making it easier to see what is going on. The **for** command syntax is

```
for initial_expression test reinit script_body
```

To use **for**, set an initial expression and then test for that condition before executing the script body. After executing the script body, the **for** command reinitializes the initial expression and again, tests for the new value, repeating the loop until the test becomes false.

continue and break Commands Terminate Loops

The **continue** and **break** commands terminate loops started with the **while**, **for**, and **foreach** commands.

Use the **continue** command to terminate the current iteration of a loop. For instance, your loop can test for, and selectively ignore particular elements in a list while continuing to operate on the rest of the elements. Use the **break** command to immediately terminate loop execution.

case Command Tests for Patterns Before Executing

Some commands return a list such as a list of objects in a directory or a list of servers running on a host system. You can use the **case** command to test a list or string for specific patterns such as the name of a particular object or server. On detecting a specified pattern, the **case** command then executes a script associated with the pattern detected. The general syntax for the **case** command is:

```
case string in pattern script pattern script
```

The **case** command looks in *string* for pattern and executes {script}. The word *in* is just a connecting word which you can omit.

If your list of patterns is lengthy and likely to break across lines, you can prevent new lines from being interpreted as separators by enclosing the entire list of target patterns and scripts in braces. This has the additional benefit of preventing variable and command substitutions in the braced list.

Patterns can include global file-name characters. A question mark (?) in a search pattern matches the corresponding character in the target pattern. For instance, **?at** matches **bat**, **cat**, and **hat**. An asterisk (*) in a search pattern matches any string in the target pattern. For instance ***at** matches **bat**, **chat**, and **"three cornered hat"** (note the use of double-quotes to disable spaces as separators).

You might want a way to execute some default script when no pattern matches are found. The **case** command has a special pattern called *default* whose corresponding script executes when no pattern match is found. You should place the default pattern as the last position in the list:

```
case $x in {
a {puts "script for case a"}
b {puts "script for case b"}
default {puts "run this script if no matches are found"}
}
```

Examples — Controlling Your Script

- If you are writing a script that searches through a list of attributes for a particular attribute, an **if** statement could take particular actions depending on whether an attribute exists. The following example script fragment returns an error message if the account name does not exist in the *list_of_group_entries* variable.

  ```
  set list_of_group_entries [group list $group -simplename]
  if { [lsearch $list_of_group_entries $account_name] == -1} {
       group add $group -member $account_name
  } else {
  error "Group \"$group\" already has an entry for \"$account_name\"."
  }
  ```

- The **foreach** command example below could be part of a script that manages hosts in a DCE cell. This script fragment removes the host principal name from the registry if a failure occurs while configuring the host in the cell. The **foreach** command looks at each principal name in the cell. If the **string** commands find the DCE host name listed in the output from **principal catalog**, the script deletes the principal name from the registry.

  ```
  foreach princ [principal catalog -simplename] {
      if {[string match $host_name [string range $princ 0 \
          [expr [string length $host_name] - 1]]] == 1} {
          principal delete $princ
      }
  }
  ```

  Keep in mind that loops return their results to the interpreter, not to **stdout**. You need to take extra steps to send the results to **stdout**. The next example uses a **puts** command to send the results of the **foreach** loop to **stdout**.

```
foreach i [group list subsys/dce/dts-servers] {
   puts [principal show $i]
    }
```

You can also append all the results together into a variable in a script, or you can use **lappend** to append the results as separate list elements.

```
foreach i [group list subsys/dce/dts-servers] {
    append result [principal show $i]
  }
  return $result
```

- The following example procedure uses a **while** loop to search through each element in a list for a pattern. As long as the list size contains more than zero elements (`$size > 0`), the procedure continues looping.

```
proc _dcp_list_find {search_list pattern} {

  set found_items ""
  set size [llength $search_list]
  while { $size > 0 } {
    set size [expr $size - 1]
    set index [lsearch $search_list $pattern]
    if { $index == -1 } {
      return $found_items
    }
    lappend found_items [lindex $search_list $index]
    set search_list [lreplace $search_list $index $index]
  }
}
```

- The following example shows a **for** loop that does an operation a specified number of times and stops. In this example, we create fifty guest principal names in the registry.

```
dcecp> for {set i 0} {$i < 50} {incr i} {principal create guest$i}
dcecp>
```

- The following example script fragment is a **foreach** command loop that includes **continue** and **break** commands. The **foreach** command looks through all the DTS servers in a cell until it finds one that is a time provider (a time provider is a special DTS server that receives time from an external time source). If the first server in the list (created by the **dts catalog** operation) returns output from a **dts show** operation, the **continue** command invokes the next lines in the script which search the output for the **{provider yes}** attribute and value. If the **provider** attribute (examined by the **attrlist getval** operation) is **yes**, the script sets the *server* variable to be the name of that DTS server and the **break** command terminates the entire **foreach** loop.

```
foreach s [dts catalog] {
    if {[catch {dts show $s} dts_sh_out] != 0} {
         continue
    }
    set p [attrlist getval $dts_sh_out -type provider]
    if {[string match $p "yes"] == 1} {
        set provider "yes"
        set server $s
        break
    }
set provider "no"
}
```

- The following examples illustrate how the **case** command works.

```
dcecp> set x {one ten twenty}
one ten twenty
dcecp> foreach el $x {case $el in one {puts script1} two {puts script2}}
script1
dcecp>
```

The **case** command first checks in *$x* variable for the pattern **one**. On finding this pattern, the associated script echoes `script 1` on the display. When it finds no more matches, the **case** command ends.

For a more practical example, say you run a **dcecp** command that lists all the servers on a particular system. You could search the list for particular server names and execute a script that appends each name to a particular file.

```
case $x in server1 {lappend filename1} server2 {lappend filename2}
```

Convenience Variable Usage

Introduction — Convenience Variable Usage

The **dcecp** program remembers what you type as well as command output, and it stores certain pieces of that information in convenience variables for reuse in subsequent commands. Using these variables in your interactive commands can reduce typing and help eliminate typing mistakes.

Convenience variables apply only to **dcecp** commands such as **directory**, **principal**, **acl**, **account**, and so on. They do not apply to Tcl commands such as **for** or **eval**, or AIX commands like **mv** or **grep**.

While this simple explanation demonstrates the general operation of convenience variables, it understates their usefulness. Most of the convenience variables are intended to aid interactive use, but some can be used in scripts as well, adding flexibility because the information they contain is not hardcoded in the script. Moreover as you gain experience with the DCE control program, you will likely find these variables to be indispensable administrative tools.

The **dcecp** program provides several convenience variables that substitute for previously typed information or command output. All of the convenience variables begin with an underscore (_) to leave one-character variable names free for other uses.

The following sections describe the convenience variables. Their order of presentation generally keeps similar or related variables together.

More Detailed Information — Convenience Variable Usage

Current Principal (User) Name (_u)

The **_u** convenience variable holds the current simple principal name. The **dcecp** program sets this variable from the login context inherited from the parent process. You can change its value by performing another **login** operation. Setting it using **set** generates an error.

```
dcecp> puts $_u
cell_admin
```

A practical use of this variable could be in scripts that test for a certain DCE identity before proceeding. On finding an incorrect identity, scripts could prompt for the necessary identity information and perform a **dce_login** operation.

Current Cell Name (_c)

The **_c** convenience variable holds the name of the cell in which the principal is registered. The **dcecp** program sets this variable from the login context inherited from the parent process. You can change its value by performing another **login** operation. Setting it using **set** generates an error.

```
dcecp> puts $_c
/.../my_cell.goodco.com
dcecp>
```

This variable is generally useful in environments where administrators deal with multiple cells.

Current Host Name (_h)

The **_h** convenience variable holds the DCE name of the current host. The **dcecp** program sets this variable when dcecp is invoked. Setting it using **set** generates an error.

The **_h** variable is useful for returning the name of the host to an interactive user. You can also use it with the **_c** variable to construct names such as a host principal name in a script.

```
dcecp> puts $_c/$_h/self
/.../my_cell.goodco.com/hosts/planets/self
dcecp>
```

Most Recent Operation Argument Name (_n)

The **_n** variable holds the name or names used as an argument to the most recent control program operation. Most DCE control program objects take a name or a list of names as an argument. Those that do not include **endpoint**, **attrlist**, **uuid**, **name**, **utc**, and the miscellaneous **dcecp** commands **dcecp_initlnterp**, **login**, **logout**, **errtext**, **quit**, **resolve**, and **shell**.

The name is usually the third argument in a **dcecp** operation as shown in the following **directory** operation.

```
dcecp> directory create /.:/sales/printers/text_printers
dcecp>
```

Note: Entering this sample command may generate an error if you have not created the account referenced in this example. Replace the variable information with information that is valid for your system.

After it is set, you can use **$_n** in subsequent operations in place of the name argument.

The **_n** variable can also hold a list of names, as when you perform a directory service operation on more than one name.

Parent of the _n(_p)

The **_p** variable holds the parent of the name stored in **_n**. The **_n** variable holds the name or list of names used in the argument to the most recent operation. The **_p** variable holds the name or list of names that are hierarchically above the name in **_n** (closer to the cell root).

One use of the **_p** variable is in traversing up a CDS hierarchy of directories. Another use is showing the access control list (ACL) of a parent object.

Last dcecp Object Name (_o)

The **_o** variable holds the name of the **dcecp** object used in the most recent operation.

Last Operation's Return Value (_r)

The **_r** variable holds the return value of the most recent operation. Many **dcecp** commands return multiple lines of output which are in the form of a list.

DCE Servers to Use (_s(xxx))

The **_s(***xxx***)** variables hold the names of the DCE servers to use for the next DCE operation. The DCE control program provides four of these variables. Because the variables are not set by the **dcecp** program, users must set these variables if they want to use them. The variables are:

_s(sec) This variable holds the name of the security server you want to use for the next registry operation. If you set this to specify a read-only replica and the operation (such as principal create) requires a master replica, the **dcecp** program ignores the variable and tries to bind to the primary registry. Registry operations that use the **_s(sec)** variable include **principal**, **group**, **organization**, **registry**, **account**, and **xattrschema**.

DCE control program operations use the **_s(sec)** variable in conjunction with the **_b(sec)** variable which holds the name of the most recent registry used. A **registry** operation uses the following order to select a security server:

1. Use the server passed as a name argument to the **registry** operation.
2. If the operation lacks a name argument, use the server named in the **_s(sec)** variable.
3. If the **_s(sec)** variable has not been set, use the server named in the **_b(sec)** variable.
4. If the **_b(sec)** variable has not been set (that is, this is the first **registry** operation since the **dcecp** program was initialized), the service provides an arbitrary server that is suitable for the operation.

_s(cds) This variable holds the name of the CDS server you want to use for the next directory service operation. When set, CDS operations attempt to use the specified server. The operation fails if the attempt is unsuccessful — such as when the server is unavailable for some reason. To overcome such a failure you must unset this variable or make the server available.

It makes sense to use the **_s(cds)** variable when all of your application needs can be satisfied by the clearinghouse named in the variable. Consider not using the **_s(cds)** variable when name lookups in CDS are likely to traverse directories in several clearinghouses. In this case, you will get lookup errors because the **_s(cds)** variable limits the lookup operation to using just the named clearinghouse.

_s(dts) This variable holds the name of the DTS server you want to use for the next time service operation. When set, DTS operations attempt to use the specified server. The operation fails if the attempt is unsuccessful — such as when the server is unavailable for some reason. To overcome such a failure you must unset this variable or make the server available.

One use of this variable is to restrict DTS operations to a single DTS server for monitoring purposes. Normally, time service operations can use any available DTS server.

_s(aud) This variable holds the name of the audit daemon you want to use for the audit operation. By default, audit operations affect the local host's audit daemon. You can operate on a remote host's audit daemon by specifying its name as the value of the **_s(aud)** variable.

```
dcecp> set _s(aud) /.:/hosts/planets/audit-server
/.:/hosts/planets/audit-server
dcecp>
```

When **S_s(aud)** is set, audit operations attempts to use the specified audit daemon. The operation fails if the attempt is unsuccessful — such as when the specified audit daemon is unavailable for some reason. To overcome such a failure you must unset this variable or make the audit daemon available.

You can specify a DCE server or audit daemon as:

- A DCE name. An example of a global registry name is **/.../my_cell.goodco.com/subsys/dce/sec/master**. An example of a cell-relative CDS clearinghouse name is **/.:/Paris_CH**.
- The string binding for the host where the server resides. String bindings can represent security servers, DTS servers, and audit daemons. They cannot represent CDS servers. An example of a string binding is `{ncacn_ip_tcp 110.15.22.131}`. The **dcecp** program resolves the binding to the appropriate service on the host.
- The name of the cell. This form applies only to **registry** operations. For a remote cell, specify a global cell name such as **/.../my_cell.goodco.com**. For the local cell you can specify the root as **/.:**. These operations use an arbitrary server that is suitable for the operation.

Last Security Server Used (_b(sec))

The **_b(sec)** convenience variable holds the name of the security server used for the most recent **registry** operation. The **dcecp** program sets this variable based on previous registry operations. Consequently, users can view, but not set, this variable.

Registry operations use the value of the **_b(sec)** variable in conjunction with the value of the **_s(sec)** variable to determine which security server to use.

Examples — Convenience Variable Usage

- The convenience variable **_n** holds the name (the argument) used in the following **principal create** operation. The **principal show** operation retrieves the name using the **$_n** variable.

```
dcecp> principal create D_Kalivas
dcecp> principal show $_n -all
{fullname {}}
{uid 17}
{uuid 00000011-d957-21cd-8d00-0000c08adf56}
{alias no}
{quota unlimited}
dcecp>
```

- You could use the **_c** variable as a building block in constructing the current context's fully-qualified principal name for use in scripts. Join the cell name and user name variables together with a slash character (/) as shown in the following example.

```
dcecp> puts $_c/$_u
/.../my_cell.goodco.com/cell_admin
dcecp>
```

- You can the **_h** variable with the **_c** variable to construct names such as in the following example:

```
dcecp> puts $_h
hosts/planets
dcecp>
```

- You could modify a directory attribute for the **/.:/sales/printers/text_printers** directory created in the preceding example.

```
dcecp> directory mod $_n -change {CDS_Convergence low}
dcecp>
```

- You could create several directories and then decide to modify an attribute.

```
dcecp> directory create {/.:/sales/printers/text_printers
                        /.:/sales/printers/graphics_printers
                        /.:/sales/printers/colorgraphics_printers }
dcecp>
```

A subsequent directory service operation can simply use the **_n** variable in place of the name or list of names:

```
dcecp> directory modify $_n -change {CDS_convergence high}
dcecp>
```

- The following operations view the ACLs of a server configuration object and of its parent object (**/.:/hosts/krypton/config/srvrconf**).

```
dcecp> acl show /.:/hosts/krypton/config/srvrconf/video_clip
{appl_admin cdfrwx}
{unauthenticated r}
{any_other r}
dcecp> acl show $_p
{appl_admin criI}
{unauthenticated r}
{any_other r}
dcecp>
```

- The following example uses the **_o** variable to avoid retyping `account`.

```
dcecp> account show j_wanders
{acctvalid yes}
{client yes}
⋮
{home /}
⋮
{shell {}}
{stdtgtauth yes}
dcecp> $_o modify j_wanders -home /.:/fs/corporate_services/users/j_wanders
dcecp>
```

- The following example shows one use of the **_r** convenience variable. The **dts show** command returns multiple lines as a list. The **attrlist getvalues** operation (see the **attrlist** reference page in the *DCE for AIX Administration Command Reference*) searches through the returned list for the string **toofewservers** and returns its associated value.

```
dcecp> dts show -counters
{creationtime 1994-09-16-07:50:13.067-04:00I-----}
{nointersections 0}
{nointersections 0}
{diffepochs 0}
{toofewservers 1}
{providertimeouts 82}
{badprotocols 0}
{badtimerep 0}
{noglobals 81}
{noresponses 0}
{abrupts 0}
{epochchanges 0}
{syserrors 0}
{syncs 1574}
{updates 0}
{enables 1}
{disables 0}
{nomemories 0}
{providerfailures 0}
{badlocalservers 0}
{badservers 0}
dcecp> attrlist getvalues $_r -type toofewservers
1
dcecp>
```

- To read the value of the **_b(sec)** variable and to identify the registry that performed the most recent operation, see the following example:

```
dcecp> puts $_b(sec)
/.../my_cell.goodco.com/subsys/dce/sec/master
dcecp>
```

Creating Commands Dynamically with the eval Command

Introduction — Creating Commands Dynamically

The **eval** command lets you create scripts as you go along by chaining smaller scripts together. This technique could be useful in a script that records administrator responses to various questions and then constructs a specialized script based on those responses. The syntax is:

```
eval arg ... arg
```

More Detailed Information — Creating Commands Dynamically

Be careful when using variables to construct **eval** commands. An **eval** command like the following can sometimes cause problems within scripts because **dcecp** parses it twice. First, **dcecp** parses the **eval** command and its arguments. Then it again parses the **eval** arguments when they are executed as scripts.

```
dcecp> eval $a $b $c
dcecp>
```

You can avoid some parsing problems by placing braces around the arguments as in:

```
dcecp> eval {$a $b $c}
dcecp>
```

To make certain the **dcecp** parses your **eval** command correctly, you can invoke the **dcecp** program's **list** command to generate a valid list structure:

```
dcecp> eval [list $a $b $c]
dcecp>
```

Example — Creating Commands Dynamically

The following example uses variables to hold options and their values for an **account create** operation. The **eval** command ensures that the variables expand and execute properly.

```
dcecp> set mpwd {-mypwd mxyzptlk}
-mypwd mxyzptlk
dcecp> set pwd {-password change.me}
-password change.me
dcecp> set org {-organization guests}
-organization guests
dcecp> set grp {-group guest}
-group guest
dcecp> eval account create guest1 $mpwd $pwd $org $grp
dcecp>
```

Note: Entering this sample command may generate an error if you have not created the account referenced in this example. Replace the variable information with information that is valid for your system.

Creating New Commands With the proc Command

Introduction — Creating Commands with proc

The **dcecp** program provides a powerful and comprehensive set of commands for controlling and monitoring DCE operations. But the exact uses to which DCE is put by end users is unpredictable. Consequently, it is quite likely that some administrators will need additional commands to meet very specific needs. The **proc** command offers an easy way to create additional commands which look and behave just like built-in commands such as **set**, **list**, and **while**. However, unlike built-in commands which are written in C, commands created with **proc** are written using scripts as in:

```
dcecp> proc div {x y} {expr $x/$y}
dcecp>
```

proc takes three arguments — the procedure name, a list of names of procedure arguments, and the **dcecp** script that forms the body of the new procedure. Our new procedure **div** requires two arguments. For example,

```
dcecp> div 12 4
3
dcecp>
```

More Detailed Information — Creating Commands with proc

By default, **proc** assumes all variables are local variables. That is, their names and values are set only within the procedure and they expire when the procedure completes. The following command produces an error because variables **x** and **y** have not been set within the procedure.

```
dcecp> set x 15
15
dcecp> set y 3
3
dcecp> proc div {} {expr $x/$y}
dcecp> div
Error: can't read "x": no such variable
```

You can import global variables (variables defined outside the procedure) using the **global** command:

```
dcecp> set x 15
dcecp> set y 3
dcecp> proc div {} {global x y > expr $x/$y}
dcecp> div
5
dcecp>
```

Note: Entering this sample command may generate an error if you have not created the account referenced in this example. Replace the variable information with information that is valid for your system.

After you import a global variable, it persists for the duration of the procedure. Your procedure can change the value of the variable using **unset** and **set**. The new value will be available for use inside and outside of your procedure.

You can use the **return** command to make your procedure return immediately. The value of the argument to **return** becomes the procedure's return value.

```
proc find {a} {
    <some pattern matching script that looks for a specific CDS entry>
    if {a != b} {
       return 1
    }
    return 0
}
```

You can design procedures to take no arguments or variable numbers of arguments. For instance a procedure with no arguments could simply perform some straightforward operation as in the following example.

```
proc _do_create_group {} {
       global rpcgroupname
       rpcgroup create $rpcgroupname
  }
```

You can also specify a default value for an argument using a nested list structure in the argument list. In the following example, the first argument **attr** must be supplied. The second argument, **value**, defaults to 1 if no argument is supplied.

```
proc _attr_show {attr {value "unset"}}  {
       puts "$attr is $value"
  }
```

Procedures can call other procedures. The current procedure can import variables from any calling procedure using the **upvar** command.

```
upvar level otherVar1 myVar1 otherVar2 myVar2
```

A **level** argument of **1** gets the variable context of the parent procedure. An argument of **2** gets the variable context of parent's parent procedure. You can also specify levels relative to the global context by preceding the **level** argument with #. A **level** of **#0** gets global variables. A **level** of **#1** gets variables from a procedure invoked from the global level.

The **otherVar** argument names the variable you want to import. You need to include the **myVar** argument to rename the variable for use in the current procedure. The following example renames the imported variable to **cargs**.

```
upvar 1 local_args cargs
```

Procedures can also execute scripts under the context of parent procedures using the **uplevel** command. This command offers a convenient way to manage your procedure's context. For example, rather than import and manipulate numerous variables from a parent procedure, use **uplevel** to connect to them all at the same time. The syntax is:

```
uplevel level arg arg arg
```

The **uplevel** command is similar to **eval**; it concatenates arguments and executes them as scripts but unlike **eval**, **uplevel** executes the script in the context specified by **level** rather than the current context. The **level** argument works the same in **uplevel** as it does in **upvar**. Use the parent's context using a **level** argument of **1**. Use the context of a first level procedure using a **level** argument of **#1**.

If a **proc** command specifies a command name that is already in effect, the new procedure replaces the existing procedure with the same name. Except in unusual cases, you should avoid naming new commands so that they replace existing built-in commands.

You can rename or delete Tcl commands using the **rename** command. For instance you could temporarily rename **list** to **list.old** and then use **proc** to create another command called **list**. When you are through using the manufactured **list** command, you could rename **list old** to **list**, restoring the original function of **list** as in:

```
rename list list.old
  proc list {} {
       <some list operation>
  }
  rename list.old list
```

Delete a command by omitting the second argument to the **rename** command. The following example deletes the list command.

```
rename list
```

Customizing dcecp Sessions

Introduction — Customizing dcecp Sessions

The **dcecp** program includes a number of commands, objects, and task scripts for performing most of the day-to-day DCE administration operations. Nevertheless, as you gain experience using the **dcecp** interface, you may find you want to add new commands and capabilities or to customize some existing ones. The following sections explain how to add scripts and new objects to your **dcecp** session. An object is just a formal implementation of a script that uses the **dcecp** help system and takes the form of *object operation*. See "DCE Control Program Usage" on page 2-15 and "Writing Scripts and **dcecp** Objects" on page 1-25 for an explanation of the fundamentals of writing **dcecp** scripts and creating new objects.

More Detailed Information — Customizing dcecp Sessions

Adding Scripts to dcecp Sessions

When you have a written script, you can make it available to one person or to everyone who is logged into the host by modifying one or more of the following files invoked when **dcecp** initializes:

[info library]/init.tcl
: This file is read first and contains standard Tcl initialization commands for the host. This affects all instances of dcecp running on a host. The file contains definitions for the Tcl **unknown** command and the **auto_load** facility used for initializing all of the **dcecp** objects. Administrators should avoid adding **dcecp** customizations to this file.

/opt/dcelocal/init.dcecp
: This file contains **dcecp**-specific startup information for the host. This affects all instances of **dcecp** running on a host. The **dcecp** scripts implementing operations and tasks are stored in the **/opt/dcelocal/dcecp** directory. Add customizations in the form of procedures to this file to make them available to all **dcecp** users on the host.

$HOME/.dcecprc
: This optional file stores user customizations which affect individual **dcecp** users (the owners of the **.dcecprc** files). Each DCE user can maintain a **.dcecprc** file and store their private procedures or alias names for operations. Modified **.dcecprc** files allow flexible administration in environments with multiple administrators. For example, different **.dcecprc** files for each administrator could use **dcecp source** commands to call in specific commands and task scripts that are tailored to particular areas of administration.

Adding New Objects to the DCE Control Program

If you have written a script as a formal **dcecp** object, you can make it available by including the new object in the same directory where other task objects reside. On AIX systems this is often **/opt/dcelocal/dcecp**. As a rule, you should add the new object to each host in the DCE cell. "DCE Administration Task Objects" on page 1-26 describes how you can use the **dcecp hostdata** object to copy scripts or other files to every host in a cell.

When you install a new script you must run the **auto_mkindex** utility to make the new object available to other users on the host. For more information about running the **auto_mkindex** utility, see “Writing Scripts and **dcecp** Objects” on page 1-25.

Example — Customizing dcecp Sessions

This section illustrates a simple task script and shows one way to make the script available for personal use. Our example begins with the control program's existing **clock** object which shows the current time. However, the time is simply a DTS timestamp from the clock on the local host as in:

```
dcecp> clock show
1994-10-03-10:22:59.991-04:00I-----
dcecp>
```

Let's say you create a procedure that gets a timestamp from a DTS server but also displays the name of the DTS server with the time as in the following example which invokes a user-created procedure called **show_clock**.

```
dcecp> show_clock
Time on mars is     1994-09-30-15:03:43.979-04:00I-----
dcecp>
```

You can make this procedure available to one user by including the procedure in the user's **.dcecprc** file. The following example **.dcecprc** includes user customizations consisting of the **_dcp_show_clocks** procedure and an alias that lets you invoke the procedure using the simpler **show_clocks** command name. Another procedure called **_dcp_whoami** shows the current login identity information. Note the order of operations in the **.dcecprc** file. Procedures are defined at the beginning of the file. Renaming and invoking the procedures must occur after the procedures are defined.

```
##
## Start up commands
##
# A simple command to rerun .dcecprc after modifications
proc .d {} {source $HOME/.dcecprc}

# Show your current login name and your current cell name.
proc _dcp_whoami {} {
  global _c _u
  return "You are '$_u' logged into '$_c'."
}

# Show the time on all of the dts servers running in your cell.
proc _dcp_show_clocks {} {
    set x [directory list /.:/hosts]
    foreach n $x {
        if {[catch {object show $n/dts-entity}] == 0} {
            set index [string last "/" $n]
            set y [string range $n [incr index] end]
            if {[catch {clock show $n/dts-entity} msg] == 0} {
                set i [expr 20 - [string length $y]]
                puts [format "Time on $y is %${i}s %s" " " \
                    [clock show $n/dts-entity]]
            } else {
                set i [expr 20 - [string length $y]]
                puts [format "Time on $y is %${i}s %s" " " \
                   "Server not responding."]
            }
        }
    }
}
```

```
# Give some procs usable names
rename _dcp_whoami whoami
rename _dcp_show_clocks show_clocks

# If I am authorized, say so
if {$_u != ""} {
  whoami
}
```

The **rename** command near the end of the file lets you invoke the **_dcp_show_clocks** and **_dcp_whoami** procedures using the easier command names **show_clocks** and **whoami**.

When you start the **dcecp** program, the last part of this file invokes the **_dcp_whoami** procedure if you are logged into DCE. If the **_u** convenience variable is set, the **_dcp_whoami** procedure prints your current login identity as:

```
% dcecp
You are 'principal_name' logged into 'cell_name'.
dcecp>
```

DCE Administration Objects

Introduction — DCE Administration Objects

A DCE cell consists of many things that need administration. As examples, CDS servers (clearinghouses), DTS clocks, and server location information are all entities in a DCE cell that require administration in one way or another. The DCE control program treats all of DCE's administrative entities as individual administration objects.

More Detailed Information — DCE Administration Objects

You operate on an entity by invoking its object name along with some operation. So for instance, to check the time of a DTS clock, you invoke the object's name (**clock**) and the desired operation (**show**) as in the following example.

```
dcecp> clock show
1994-09-23-10:46:42.016-04:00I-----
dcecp>
```

Each administrative entity in DCE has a corresponding administration object in the DCE control program. As a few examples, you can manage CDS clearinghouse operations in a cell using the **clearinghouse** object. Manage application servers and their configuration information on DCE hosts using the **server** object. Compare and manipulate time information using the **utc** object. Administer users in a DCE cell with the **user** object. These examples represent just a few of the **dcecp** administration objects. All of the objects are listed in the **dcecp** reference page of the *DCE for AIX Administration Command Reference*.

DCE Control Program Usage

Introduction — DCE Control Program Usage

This section provides a quick look at how to start and stop the DCE control program, how to perform operations, and describes the commands that the DCE control program provides specifically for managing the Security Service. Additional information about these topics is provided in the **dcecp** reference pages.

More Detailed Information — DCE Control Program Usage

The DCE control program creates and maintains principals, groups, organizations, and accounts for the Security Service's network-wide registry (Registry Service component). The control program also operates on the keytab files that protect the passwords for Security servers on the local node (Authentication Service component). Additionally, it maintains the ACLs that protect DCE resources (Privilege Service component). The DCE control program commands for managing the Security Service operate on these Security Service and DCE-wide resources through various objects that it defines. For example, the control program's **acl check** command displays the permissions that the ACL for a Security Service object grants to the invoking principal.

Starting and Stopping the dcecp Program

You can enter **dcecp** operations directly from your operating system prompt or from within the DCE control program. If you are performing just one or two simple **dcecp** operations, you can invoke them directly at the operating system prompt.

If you will be doing several operations, you can invoke the DCE control program and then enter operations at the **dcecp** prompt. This method offers several advantages.

- It is more efficient for multiple operations because **dcecp** is initialized once rather than for each separate operation.
- The program stores operations in a history facility so they can be recalled and reused.
- You avoid the extra keystrokes needed to precede each operation with the **dcecp** command.

When you are through using the DCE control program, use the **exit** or **quit** operation to stop the program and return to the operating system prompt. The following example illustrates using the **exit** operation.

```
dcecp> exit
%
```

All **dcecp** object, operation and option names can be abbreviated to the shortest unique string when used interactively. These names have been chosen with this in mind so that unique abbreviation are usually not more than one or two characters.

Avoid using object or command abbreviations within scripts as this limits a script's portability. Users defining their own commands could alter the uniqueness of abbreviations, resulting in ambiguous command names or object names.

Security Service Objects

The DCE control program has functions that operate on the following Security Service components:

principal This object represents Registry principals. These principals can be human users of the network, servers on the network, machines on the network, or cells with which the local cell will engage in cross-cell authentication. Optionally, the principal object can be Double-Byte Character Set (DBCS) enabled.

group This object represents Registry groups. Groups are collections of principals for which you can assign access rights to objects. Optionally, the group object can be DBCS enabled.

organization This object represents Registry organizations. Organizations are collections of principals to whom you can assign policies that expand your areas of administrative control. Optionally, the organization object can be DBCS enabled.

account This object represents the accounts that are established in the Registry for principals.

registry This object represents the Registry, or Security Service's database of account information, in a DCE cell. The Registry copy operated on can be either the primary replica or a secondary replica.

xattrschema This object operates on the schemas, or the definitions, for Extended Registry Attributes (ERAs) that you specify for Security Service components and data maintained by the Host daemon (**dced**) on the local host.

acl This object represents the ACLs for all of the DCE entities that can be protected by the ACL facility of the Security Service.

keytab This object represents the files that store the keys, or passwords, for authenticated server principals in the Security Service.

Security Service Command Operations

The DCE control program's commands for managing the Security Service can perform the operations summarized in the following list:

Commands	Operations
add	Adds a principal to a group or organization in a Registry replica.
catalog	Displays the names of all the principals, groups, and organizations in a Registry replica. For the Registry itself, displays the primary and secondary replicas existing in a DCE cell.
check	Displays the permissions that a DCE ACL currently grants to a Security principal.
create	Creates a new principal, group, organization, or account in a Registry replica. Also, creates a new entry for an ERA schema.
delete	Deletes a principal, group, organization, or account from a Registry replica. For the Registry itself, deletes a secondary replica. For an ERA schema, deletes entries. For a DCE ACL, removes ACL entries.
disable	Disables the primary replica of the Registry for updates.
dump	Displays information on each replica of the Registry existing in a cell.
enable	Enables the primary replica of the Registry for updates.
help	Displays help information about a principal, group, organization, account, ERA schema, or DCE ACL in a Registry replica, or about the Registry replica itself.
list	Displays the names of the principals belonging to a group or organization in a Registry replica.
modify	Modifies the attribute information in a Registry replica for a principal, group, account, ERA schema entry, DCE ACL entry, or for the Registry itself. For an organization, also modifies the policy information.
operations	Displays the operations that can be performed by or on a principal, group, organization, account, ERA schema, DCE ACL, or Registry replica.
permissions	Displays the permissions granted by an ACL on a protected DCE component.
remove	Removes one or more principals from a group or organization in a Registry replica.
rename	Changes the name of a principal, group, organization, or ERA schema in a Registry replica.
replace	Replaces the entire ACL on a DCE component.
show	Displays information about the attributes of a principal, a group, ERA schema entries, or DCE ACL entries. Also displays information about the policies for an organization, account, or Registry replica.

stop Stops a Security server process.

synchronize Instructs the secondary replica of the Registry to update its contents from the primary replica.

verify Checks if all of the Registry's replicas are up to date.

Specific instructions for using DCE control program commands to create and maintain principals, groups, organizations, and accounts are given in “Creating and Maintaining Principals, Groups, and Organizations” on page 1-48 and “Creating and Maintaining Accounts” on page 1-48.

Examples — DCE Control Program Usage

- The following example shows how to invoke the DCE control program and perform a **directory** operation.

```
% dcecp
dcecp> directory create /.:/hosts/appserver2
dcecp>
```

- If you are performing a single **dcecp** operation, you can invoke it directly from your operating system prompt. Just precede the desired operation with the **dcecp** command and the **–c** (command line operation) argument at the operating system prompt:

```
% dcecp -c directory list /.:/subsys -simplename
HP applications dce sales eng admin accts
% dcecp -c cell show
{secservers
 /.../my_cell.goodco.com/subsys/dce/sec/master}
{cdsservers
 /.../my_cell.goodco.com/hosts/krypton}
{dtsservers
 /.../my_cell.goodco.com/hosts/mars}
{hosts
 /.../my_cell.goodco.com/hosts/earth
 /.../my_cell.goodco.com/hosts/jupiter
 /.../my_cell.goodco.com/hosts/kyrpton
 /.../my_cell.goodco.com/hosts/mars
 /.../my_cell.goodco.com/hosts/mercury
 /.../my_cell.goodco.com/hosts/neptune
 /.../my_cell.goodco.com/hosts/pluto
 /.../my_cell.goodco.com/hosts/saturn
 /.../my_cell.goodco.com/hosts/uranus
 /.../my_cell.goodco.com/hosts/venus}
%
```

- You can also enter some limited multiple operations using the semicolon (;) as a command separator and enclosing the operations in double quotes. The following example adds a principal to the registry and then checks that the principal is added.

```
% dcecp -c "principal create S_Preska ; principal show S_Preska"
{fullname {}}
{uid 28}
{uuid 0000001c-dc77-21cd-b700-0000c08adf56}
{alias no}
{quota unlimited}
%
```

- Be careful entering multiple operations using the **dcecp** command with the **–c** option because operation results return to the **dcecp** interpreter, not to the shell. An operation like the following returns the results of just the last operation (**group list users**) to the shell.

```
% dcecp -c "group list staff; group list managers; group list users"
/.../ward_cell.osf.org/P_Pestana
/.../ward_cell.osf.org/R_Parsons
/.../ward_cell.osf.org/L_Jones
/.../ward_cell.osf.org/S_Preska
/.../ward_cell.osf.org/N_Long
/.../ward_cell.osf.org/D_Witt
/.../ward_cell.osf.org/C_Pilat
⋮
%
```

- To invoke a **dcecp** script, omit the **–c** argument but include the name of the script. The following example invokes a script that lists the names of all hosts in the cell in alphabetical order.

```
% dcecp list_hosts
earth
jupiter
krypton
mars
mercury
neptune
planets
pluto
saturn
uranus
venus
%
```

- When you want to invoke complex or multiple operations, you might want invoke operations from within the **dcecp** program. The program provides a convenient history facility and a command line editing capability that is useful for recalling and reusing previous operations. The following example operations invoke the **dcecp** program and add a new user to a DCE cell.

```
% dcecp
dcecp> principal create J_Jones
dcecp> group add users -member J_Jones
dcecp> organization add staff -member J_Jones
dcecp> account create J_Jones -group users -organization staff \
> -password change.me -mypwd mxyzptlk
dcecp>
```

dcecp Commands Usage

Introduction — dcecp Commands Usage

The DCE control program accepts commands ranging from simple to complex, with more complex commands offering greater strength and versatility. Although simple commands are the easiest to compose, they are also limited, usually to performing one operation on a single object. So while it is always possible to enter simple commands, you will probably find that at times, you want to repeat operations over several or even many objects, or to perform some operation only under certain conditions. For instance you might want to add some entry to a CDS directory only if some other specified entry already exists in CDS. The **dcecp** program makes this possible by utilizing Tcl's built-in commands that imitate elements commonly found in numerous programming and shell languages.

More Detailed Information — dcecp Commands Usage

The **dcecp** program contains many C-like constructs that control command execution. Some examples are if statements for conditional execution, looping commands such as **while**, **for**, and **foreach** used to repeat operations under various conditions, a **case** command for testing values against various patterns, and **proc** for writing your own customized commands.

The **dcecp** program also includes other syntactic elements such as quotes (“”), braces ({}), brackets ([]), and the backslash (\) character which it uses to group elements together and for controlling interpretation of special characters.

Although many features are designed for use in scripts, you will probably find yourself using some constructs and elements (particularly quotes, braces, brackets, and backslashes) in interactive operations as well. You will need to decide when it makes sense to perform operations interactively or to use a script. In general, complexity and potential for reuse can help you decide.

Braces also help **dcecp** determine whether a command is complete; incomplete commands will have more opening than closing braces. The lack of a closing brace at the end of the first line signals **dcecp** that more command input is coming so **dcecp** prompts with the secondary prompt (>). Similarly, the opening brace at the end of line 2 signals that you are still not finished entering the command. This lets you wrap lines without using a backslash (\) line wrap character. The **dcecp** program executes the command when you press <Enter> after the closing brace at the end of line 3. “DCE Control Program Language Usage” on page 1-24 contains more information about braces.

This is a high-level look at some practical uses of **dcecp**. Of course there are many more uses which we have not covered. See “DCE Control Program Language Usage” on page 1-24 for more information on some of the **dcecp** operations that you are likely to use for DCE administration. Remember that **dcecp** is based on Tcl and Tcl has other commands and command variations we will not discuss. So be sure you have access to the standard Tcl publications for detailed information on all of the commands.

Examples — dcecp Commands Usage

- Here are some simple examples that illustrate some DCE control program and Tcl basics. Some **dcecp** operations can be very straightforward like

  ```
  dcecp> account modify N_Long -expdate 1996-06-30
  dcecp>
  ```

  This operation lets you change information in the DCE Security Service registry. Here, we are changing the account expiration date for the principal (N_Long) named in the command line. While it is relatively simple to execute this operation for one or two principals, it is more difficult to change the account expiration date for many principals. Imagine that your organization employs six temporary workers and the project they are associated with has been extended for three months. Rather than execute the **account modify** operation six times, you can use a **dcecp foreach** command to loop (repeat) an action for each item of a list:

  ```
  dcecp> foreach i {N_Long L_Jones P_Sawyer
                    D_Witt M_Dougherty S_Preska}
                    {account modify $i -expdate 1996-06-30 }
  dcecp>
  ```

  In the example, the **foreach** looping command has three arguments; a variable, a list, and the body. The variable **i** substitutes sequentially for each item in the list (N_Long, L_Jones, and so on). The **foreach** command executes the body (**account modify $i –expdate 1996-06-30)**) for each item in the list. **$i** in the body takes on the value of each principal name in the list, in turn, until all items in the list have been used. See “Controlling Your Script” on page 2-1 for more detailed information about looping commands.

This example illustrates several other important syntax rules. The **dcecp** program uses braces ({}) to determine where command arguments, such as the script body, begin and end. For example, the **foreach** command has three arguments: a variable name, a list, and a script body. Normally command arguments are separated by spaces. To prevent **dcecp** from incorrectly interpreting the spaces between list elements as argument separators, we use braces to enclose the list and disable special interpretation of the spaces. Thus all of the list elements appear as one argument. Similarly we use braces to enclose the individual elements in the script body.

- Now, assume that instead of six temporary workers, your organization has fifty temporary workers (all in one group called **temps**) for whom you want to add three-month account extensions. We will still use the **foreach** command but rather than write all fifty principals directly in the list, use the **dcecp group list temps** operation to generate a list for you.

  ```
  dcecp> foreach i [group list temps] {account modify $i -expdate 1996-06-30 }
  dcecp>
  ```

  In this example, we put the **group list temps** operation in square brackets ([]). This technique (called *command substitution*) replaces the command inside the square brackets with the results returned by that command. The results of the **group list temps** operation produces a valid **Tcl** list that might look like:

  ```
  dcecp> group list temps
  N_Long
  L_Jones
  P_Sawyer
  D_Witt
  M_Dougherty
  S_Preska
  ⋮
  J_Jones
  ```

dcecp Help Facilities Usage

Introduction — dcecp Help Facilities Usage

The DCE control program offers help in several ways:

- If you want to see a list of objects provided by the DCE control program, enter `help` at the **dcecp** prompt.
- If you just need to know which operations an object supports, use the *object* **operations** command. This command returns a list of the actions you can take on an object.

  You can save typing by abbreviating this command to something like **prin oper**.
- Get more detailed help about an object and its operations by using the *object* **help** command.
- Get information about available command options by adding an *operation* argument to the object **help** command.
- Get help about an object itself by using an *object* **help –verbose** command.
- Finally, some POSIX style systems will have manpages for **dcecp** objects as well as a Tcl summary manpage. Each **dcecp** object has its own manpage which describes the object and the operations available to it. The general syntax for viewing a **dcecp** object manpage is:

  dcecp> **man** *object_name*

  The **Tcl** manpage summarizes the Tcl built-in commands. You can view the Tcl summary manpage on a UNIX style system by entering:

```
dcecp> man Tcl
⋮
dcecp>
```

Examples — dcecp Help Facilities Usage

- To see a list of objects provided by the DCE control program, enter `help` at the **dcecp** prompt, as follows:

```
dcecp> help
The general format of all dcecp object operations is as follows:
dcecp> <object> <verb> [argument] [options]

In addition to all of the standard tcl commands, dcecp supports many commands
to administer DCE objects.  A dcecp object or task represents a DCE entity.  All
of the following dcecp objects and tasks require a verb:
account       cdscache        group          object           secval
acl           cell            host           organization     server
attrlist      clearinghouse   hostdata       principal        user
aud           clock           keytab         registry         utc
audevents     directory       link           rpcentry         uuid
audfilter     dts             log            rpcgroup         xattrschema
audtrail      endpoint        name           rpcprofile

Miscellaneous commands perform specific functions.  Type 'man <command>' for
more information.  These commands take no verb:
echo     errtext  login    logout   quit     resolve  shell

To list all dcecp objects:                    dcecp> help -verbose
To list all verbs an object supports:         dcecp> <object> help
To list all options for an object operation:  dcecp> <object> help <verb>
For verbose information on a dcecp object:    dcecp> <object> help -verbose
For the manual page on a dcecp object:        dcecp> man <object>
dcecp>
```

- The following example shows how to list the operations available for the **principal** object.

```
dcecp> principal operations
create delete modify show catalog operations help
dcecp>
```

- The following example returns a one-line description of each operation supported by the **principal** object.

```
dcecp> principal help
catalog         View a list of all the names of principals in the registry
create          Create a dce principal
delete          Delete a principal from the registry
modify          Change the information about a principal
rename          Rename the specified principal.
show            Returns an attribute list of the specified principal.
help            Print summary of command-line options and abort.
operations      Return valid operations for command.
dcecp>
```

- The following example returns a one-line description of each option supported by the **principal create** operation.

```
dcecp> principal help create
-alias          Add principal named as an alias of specified uid
-attribute      Attribute list to be assigned to the principal to be added
-fullname       Fullname of the principal to be added
-quota          Quota of the principal to be added
-uid            User Identifier of the principal to be added
-uuid           Orphaned UUID to be adopted by the specified principal
dcecp>
```

- The following example returns a description of the **principal** object along with information about how to use the object.

```
dcecp> principal help -verbose

The principal object represents Registry principals.  All of
the principal operations take one argument which is a list of
names of principals to act on.  It must be a principal
name, not the name of the database object that contains the
registry information about the principal (i.e., it should
not begin with /.:/sec/principal/).  After this operation
executes, the s(sec) convenience variable is set to the name
of the server that was bound to for the operation.  The value
of the variable before the operation is treated as a hint.
In the registry, each principal has a primary name and a
value for each of the following attributes: alias, fullname,
quota, uid, uuid.
dcecp>
```

- The following example shows how to invoke the manpage for the **principal** object. Note that you can use the **man** command from within the **dcecp** program.

```
dcecp> man principal
⋮
dcecp>
```

Documenting Your Scripts with Comments

Introduction — Documenting Your Scripts

When you are writing scripts, you might want to include some comment lines to remind yourself and others what the script is doing. Use the number symbol (#) to insert comments. The **dcecp** program suppresses interpretation between a number symbol and the next new line. You must place the number symbol in a position where **dcecp** expects the first character of a command.

Examples — Documenting Your Scripts

- Both of the following examples are valid:

```
set a 5
# sets a to 5

set a 5 ;# sets a to 5
```

- The following example is not valid because the number symbol is not positioned where **dcecp** expects the first character of a command.

```
set a 5 # sets a to 5
```

- A common use of comments is to document procedures in scripts as in the following example script fragment.

```
⋮
#
# _dcp_cleanup_user_create  - This function undoes changes after a
# failure in one of the user create functions as though the operation
# never occurred.
#
proc _dcp_cleanup_user_create {account_name args} {
⋮
```

Editing Command Lines

Introduction — Editing Command Lines

The **dcecp** program offers several ways to edit commands. You can edit a current command line using the command line editing facility. You can use the **history** command to recall, edit, and reissue a previously used command.

More Detailed Information — Editing Command Lines

Editing the Current Command Line

You can edit a command line before sending it to the *dcecp* by typing control characters or escape sequences that resemble **ksh** or **emacs** editing commands. A control character, shown as a <Ctrl-> followed by a letter, is typed by holding down the control key while the letter is typed. For example, <Ctrl-A> is a control-A. Enter an escape sequence by pressing the escape key followed by one or more characters. The escape key is abbreviated as <Esc>. Case matters in escape sequences (unlike control keys which don't distinguish between upper and lower case; <Esc-F> is not the same as <Esc-f>.

You can type an editing command anywhere on the line, not just at the beginning. In addition, a return may also be typed anywhere on the line, not just at the end.

Most editing commands accept a repeat count, *n*, where **n** is a number. Enter a repeat count, by pressing the escape key, the number, and then the command to execute. For example, <Esc><4><Ctrl-f> moves forward four characters. Some of the descriptions that follow include the text [n] to identify commands that accept a repeat count.

The following control characters are accepted:

<Ctrl-A>	Move to the beginning of the line
<Ctrl-B>	Move left (backwards) [n]
<Ctrl-D>	Delete character [n]
<Ctrl-E>	Move to end of line
<Ctrl-F>	Move right (forwards) [n]
<Ctrl-G>	Ring the bell
<Ctrl-H>	Delete character before cursor (backspace key) [n]
<Ctrl-I>	Complete file name (tab key); see below
<Ctrl-J>	Done with line (return key)
<Ctrl-K>	Kill to end of line (or column [n])
<Ctrl-L>	Redisplay line
<Ctrl-M>	Done with line (alternate return key)
<Ctrl-N>	Get next line from history [n]
<Ctrl-P>	Get previous line from history [n]
<Ctrl-R>	Search backward (forward if [n]) through history for text; must start line if text begins with an up arrow
<Ctrl-T>	Transpose characters
<Ctrl-V>	Insert next character, even if it is an edit command
<Ctrl-W>	Wipe to the mark
<Ctrl-X><Ctrl-X>	Exchange current location and mark
<Ctrl-Y>	Yank back last killed text
<Ctrl-[>	Start an escape sequence (escape key)
<Ctrl-c>	Move forward to next character (c)
<Ctrl-?>	Delete character before cursor (delete key) [n]

The following escape sequences are provided.

<Esc> <Ctrl-H>	Delete previous word (backspace key) [n]
<Esc> <Del>	Delete previous word (delete key) [n]
<Esc> <Space>	Set the mark (space key); see <Ctrl-X><Ctrl-X> and <Ctrl-Y> above
<Esc> <.>	Get the last (or [n]'th) word from previous line
<Esc> <?>	Show possible completions; see below
<Esc> <<>	Move to start of history
<Esc> <>>	Move to end of history
<Esc> <b>	Move backward a word [n]
<Esc> <d>	Delete word under cursor [n]
<Esc> <f>	Move forward a word [n]
<Esc> <l>	Make word lowercase [n]
<Esc> <u>	Make word uppercase [n]
<Esc> <y>	Retrieve back last killed text
<Esc> <w>	Make area up to mark retrievable
<Esc> <nn>	Set repeat count to the number *nn*

The *dcecp* program also provides file name completion. Suppose the root directory has the following files in it:

```
bin       vmunix
core      vmunix.old
```

If you type `rm /v` and then press the tab key, the command processor completes as much of the name as possible off by adding **munix**. Because the example name is not unique, it beeps. If you type the escape key and a question mark, it displays the two choices. The command processor completes the file name when you then type the period (which makes the name unique) followed by a tab:

rm /v[TAB]*munix***.[TAB]***old*

<Tab> indicates the tab key, italics indicates text automatically entered by the command processor.

Editing Command Lines With the history Command

Sometimes when you're entering interactive commands, you want to recall and reuse a previously entered command. Let's say you list the objects in a CDS directory and then you modify one of the objects. Now you want to list the objects again to verify that your modification took effect. You can use the **history** command to recall, edit, and reissue a previously used command. The history facility saves only interactive commands. Commands issued from scripts aren't saved and can't be recalled.

The **history** command takes various arguments depending on what you want to do. Entering **history** with no arguments lists all the commands (called events) entered during the current invocation of *dcecp*.

```
dcecp> history
     1  principal create wardr -fullname {Ward Rosenberry} -quota unlimited
     2  group add users -member wardr
     3  organization add consultants -member wardr
     4  account create wardr -mypwd mxyptlk -password qwerty -group users\
         -organization consultants
dcecp>
```

Each history event is independent of previous events. This means if a recalled command used a variable, its current value may not be the same as when it was first entered. The **history** command itself generates a history event, too.

By default, the history list keeps the 20 most recent commands. You can use the **history keep** command to lengthen or shorten the history list. For example the following command lengthens the history list to keep the 50 most recent events:

```
dcecp> history keep 50
dcecp>
```

You can specify events in various ways. Positive numbers specify events relative to the earliest event in the list. Negative numbers specify events relative to the most recent command. You can also specify an event by typing characters that match all or part of a previous event.

Examples — Editing Command Lines

- The history facility lets you reuse previous events in many ways. The following examples show just a few of the history commands you can use.
  - Execute a previous command without revision using the **history redo** command.

    ```
    dcecp> history
               1 directory show /.:/printers
               2 object create /.:/printers/ascii_printer1
               3 object create /.:/printers/ascii_printer2
               4 object create /.:/printers/ascii_printer3
    dcecp> history redo directory
    directory show /.:/printers
    ⋮
    dcecp>
    ```

  - You can save the most typing by entering just the unique first characters of words in a history command. For instance, you can enter the **history redo directory** command from the previous example as:

    ```
    dcecp> hi r d
    directory show /.:/printers
    ⋮
    dcecp>
    ```

    Other ways to redo commands include **!!** which recalls the most recent command and !*event number* to recall a specific event.

  - You can revise and re-execute a previous command using the **history substitute** command. The most common use of this command might be to correct typing mistakes. It's syntax is:

    history substitute *old new (event number)*

- If you omit the *event number*, you will redo the most recent command. Replace the old part of the recalled command with new information:

  ```
  dcecp> history
             1 directory show /.:/printers
             2 object create /.:/printers/ascii_printer1
             3 object create /.:/printers/ascii_printer2
             4 object create /.:/printers/ascii_printer3
             5 directory show /.:/printers
  dcecp> hi s -2 printer3 printer4
  object add /.:/printers/ascii_printer4
  dcecp>
  ```

- You can recall and revise the most recent command using the *[old[new* syntax familiar to users of the UNIX **csh** shell as in:

  ```
  dcecp> ^4^5
  object add /.:/printers/ascii_printer5
  dcecp>
  ```

Error and Exception Handling

Introduction — Error and Exception Handling

The **dcecp** interpreter includes error facilities that return error information when something goes wrong with a **dcecp** script. Error information tells users what went wrong so that they can avoid making the same mistake in the future. Many things can cause **dcecp** errors. For instance, a command might not receive the correct number of arguments, a command might have a typographic error of some kind, or the object of an operation, such as a CDS directory, might be unavailable for some reason.

More Detailed Information — Error and Exception Handling

Using Global Error Information Variables

When **dcecp** encounters an error, it prints an error message describing the error such as:

```
dcecp> set x y z
Error: wrong # args: should be "set varName ?newValue?"
dcecp>
```

When errors occur while a complex part of the script is executing, such as during a nested procedure call, error messages alone might be insufficient for determining exactly where your problem occurred. So **dcecp** stores additional error information in a global variable called *errorInfo*. Your script can access this information and print it to help you pinpoint the error. Generally, this extended error information traces the commands that were executing when the error occurred.

Using Catch to Trap Errors and Exceptions

Occasionally, you might want to trap some kinds of errors rather than let them terminate an active command. The **catch** command lets you trap and ignore errors so your script can continue processing. For example, if your script wants to rename a command if it exists. However, it is possible that the command name might not exist when you execute the rename command.

```
dcecp> rename move move.old
Error: can't rename "move":  command doesn't exist
dcecp>
```

Use **catch** to invoke the **rename** command as a script.

```
dcecp> catch {ren move move.old}
1
dcecp>
```

The **catch** command treats its argument as a script and executes it, returning a **0** on successful execution. If an error occurs, it is caught by the **catch** command which returns a **1**.

You can add a second argument to the **catch** command. This argument is a variable which **catch** modifies to hold the script's return value (on successful completion) or the error message. The syntax for the **catch** command is

```
catch command varName
```

One use of **catch** in scripts is to invoke other procedures. You can read the following script fragment as follows:

> If the **_dcp_create_group** procedure returns unsuccessfully (!= 0) then do the **_dcp_cleanup_user_create** procedure and display the error stored in the **msg** variable.

```
if {[ catch {_dcp_create_group $group group_created} msg] != 0 } {
          _dcp_cleanup_user_create $element -principal
          error $msg
    }
```

Exceptions are a special class of error generated by the **break**, **continue**, and **return** commands. You use the **break** and **continue** commands to terminate loops such as **while**, **for** and **foreach**, and you use the **return** command to terminate a **proc** or **source** command.

Resulting exceptions can be hard to handle in procedures where loops exist inside (as part of) a more comprehensive command. For instance, a user-written procedure that searches for specific object types in CDS might invoke **foreach** as part of a looping activity to test for the occurrence of particular attributes.

If you use the **break**, **continue**, or **return** commands to manage loop execution or to manage some other nested command (like **case**, or **if**, for example), the parent command will not be ready to catch the exception. The parent command will abort and issue an error message as usual. However, the error is associated with the parent command and is difficult to track to the looping command where it actually occurred.

If it is necessary to use a **continue**, **break**, or **return** command to terminate a command that has been called by another command, consider using **catch** to invoke the nested command which, in turn, calls the **continue**, **break**, or **return** command to recover from errors or exceptions. Used this way, the **catch** command keeps the exception within the looping or nested procedure where it is easier to track down.

```
foreach s [server catalog] { if {[catch
{server show $s} srv_sh_out] != 0} {
continue }
```

Reissuing Complex Errors

The **proc** command lets you create procedures or commands that perform very precise operations. For instance, a user-written procedure called **_dcp_get_servers** that retrieves and filters information about running servers could include nested commands or procedures that perform various subtasks such as looping through server information looking for certain strings. While use of nested commands or procedures lets you develop comprehensive procedures or commands, they can also produce errors that are difficult to pinpoint if errors are not passed along properly.

Complex scripts can use the **error** command to reissue errors that have been triggered by some previously executing part of the script. The following script fragment simply prints out a hard-coded error message. This use also lets you custom-tailor messages to precisely explain error conditions.

```
set dts_cat_out [_dcp_dts_catalog]
        if {[llength $dts_cat_out] == 0} {
            error "Unable to find any DTS servers"
        }
```

The next script fragment does more, using **catch** to store any error information returned from the **_dcp_create_group** procedure in the *msg* variable. On failure (!= 0), the script invokes a cleanup procedure that undoes whatever was done, and then prints out the message stored in the *msg* variable.

```
if {[ catch {_dcp_create_group $group group_created} msg] != 0 } {
              _dcp_cleanup_user_create $element -principal
              error $msg
        }
```

This discussion has provided some fairly simple error handling techniques. Note though, that error handling can be complicated, especially in more complex situations. We encourage you to read more about error handling in other publications that cover more general use of Tcl.

Example — Error and Exception Handling

The following command example shows the kind of information that can be stored in the *errorInfo* variable. Reading backwards you can reasonably determine that the error occurred near line 4 of the script body in the **parseagrs** procedure called from the **_dcp_create_user** procedure of a **user** operation.

```
dcecp> puts $errorInfo
Unknown option "group"
    while executing
"
    invoked from within
"
    ("while" body line 4)
    invoked from within
"
    (procedure "parseargs" line 60)
    invoked from within
"
    (procedure "_dcp_create_user" line 64)
    invoked from within
"
    invoked from within
"
    invoked from within
"
    (procedure "user" line 24)
"
dcecp>
```

Formal Task Objects

Introduction — Formal Task Objects

Some DCE environments might have special administration needs that aren't strictly addressed by the standard DCE control program objects. While you could write and distribute informal scripts to meet this administration need, you would likely need to document their operation in some way. More importantly, though, a complicated operation might require the use of numerous options to precisely control the script's behavior. Rather than invent your own mechanisms to provide help information and handle complicated argument parsing operations, you could rely on the existing help system and the **parseargs** facility utilized by other formal task objects supplied with the **dcecp** program. This approach makes your script consistent with other **dcecp** objects.

More Detailed Information — Formal Task Objects

Formal task objects build on the idea of the informal scripts presented previously with some important additions:

- An argument table at the beginning of the script defines operations as separate procedures within the script. An argument table can also define available options. A **parseargs** procedure is called to parse the arguments and options passed to the script when it is invoked.
- Help information for each operation is placed in the argument tables in the script. Other script users can get this information using standard **dcecp help** operations.
- Extensive error control is included because you cannot predict or control the conditions in which the script executes.

The remainder of this section shows the general structures and conventions used in a formal task object. To aid our explanation, we use the **dcecp user** task object supplied with the DCE control program.

A Model for Task Objects

This section examines the parts of the **user** task object that should be emulated in other task objects that you create for use with the DCE control program. Adhering to the basic model ensures that your task object will look and behave consistently with other parts of the **dcecp** program.

For efficiency and readability, the example does not include all of the procedures contained in the **user** task object. Furthermore, we have omitted some repetitive parts of the included procedures, replacing the omitted parts with vertical ellipses in the code examples. The entire **user** task object is contained in **/opt/dcelocal/dcecp**.

Name your object after the entity on which it operates rather than as a verb such as show or modify. DCE control program objects are named for the DCE entity on which they operate. Primitive objects like **rpcentry** and **principal** objects operate on single manageable DCE entities. Task objects operate at a higher level, generally invoking several primitive objects to achieve their goal. The authors of the user task object contrived a higher level entity — a user — as a manageable object.

The **user** object begins with the top level **proc** command and its argument table that defines the procedures and operations provided by the **user** object. Use the following syntax to define separate procedures in this argument table:

verb **command function_call** *procedure_name* **"***helptext_string***"**

The call to the **parseargs** procedure (defined in a separate file called **parseargs.dcp**) returns the name of the internal procedure that is to be called along with its arguments.

```
# proc user - This procedure is the front end for the user task
# scripts. All argument checking for the provided switches is done
# in the individual functions.
#
proc user { args } {
  set arg_table {
    {create command function_call _dcp_create_user
       "Create a DCE user" }
    {delete command function_call _dcp_delete_user
       "Delete a DCE user"}
    {show   command function_call _dcp_show_user
       "Show the attributes of a DCE user"}
    {help   help  help_list
       "Print summary of command-line options and abort"}
    {operations operations operation_list
       "Return valid operations for command."}}
  set verbose_prose
"This object allows the manipulation of a DCE user.  A user is
represented as a principal and account with membership in a group and
organization as well as having a directory in the CDS namespace.  A user
may be created, deleted or have attribute information returned.  The
argument is a list of either relative or fully qualified principal names.
All fixed attributes of the principal and account object may be specified
when creating a user.  The -force option to the create verb allows the
group or organization for that user to be created if necessary.  The user
is provided a directory in the CDS namespace, with the appropriate ACLs.
Access to create a user requires the correct ACLs on principal, group and
organization directories within the registry and the clearinghouse and
users directory in the CDS namespace."

  set local_args $args
```

```
  parseargs $arg_table local_args -found_one

  if { [info local help_prose ] > 0 } { return $help_prose }
  if { [info local function_call ] > 0 } {
    return [$function_call local_args]
  } else {
    error "\"user\" object requires a verb to form a command."
  }
}
```

The next part of the script examines a procedure that takes many options or attributes as input — **_dcp_create_user**. While this procedure relies on numerous lower-level procedures to do the actual work of creating a user, the example begins by showing just one of the lower-level procedures; **_dcp_create_principal_entry**.

Then the script continues with the **_dcp_create_user** procedure. Notice that the name of this procedure (and all lower-level procedures) begins with an underscore. That's because the Tcl **info** command is frequently used to return the names of all procedures. This convention distinguishes these internal procedure names from procedures like **user** which are documented procedures. Furthermore, the **_dcp** part of the name distinguishes **dcecp** procedures from other Tcl procedures on a host.

The **_dcp_create_user** procedure has an argument table defining its available options. This argument table differs from the script's initial argument table in that it lacks the **command** keyword and the *function_call* variable that define separate procedures in the script.

Next it initializes variables entered either as options or as attributes in a list. A **process_attribute_list** procedure (at the end of the example) actually parses attributes that have been passed as a list. Then it does the work of creating the user information in the registry and in CDS. Near the end, a cleanup procedure **_dcp_cleanup_user_create** can undo a failed user create operation.

```
⋮
#
# This procedure creates a principal in the current registry _s(sec)
# if that principal does not yet exist.
#
proc _dcp_create_principal_entry { principal_name princ_args} {
  set list_of_principals [principal catalog]
  if { [lsearch $list_of_principals $principal_name] == -1} {
    if { [llength $princ_args ] != 0 } {
      principal create $principal_name -attribute $princ_args
    } else
  } else {
    error "Principal \"$principal_name\" already exists."
  }
}
#
# proc _dcp_create_user - This procedure actually creates a DCE user.
# Several steps are performed. If the principal does not exist
# a new one is created. If the groups do not exist and a -force switch is
# set, then two new groups will be added. The user will be added to the
# groups. The account will then be created. An entry in the CDS
# namespace will then be created with the appropriate ACL's.
#
proc _dcp_create_user { local_args } {
  set arg_table {
    {-alias string alias
          "Add principal named as an alias of specified uid."}
    {-attribute string attribute_list
          "Provide attributes in an attribute list format."}
    {-client string client
          "Can the account principal be a client."}
```

```
    {-description string descr
           "A general description of the account."}
    {-dupkey string dupkey
           "Can the accounts's principal have duplicate keys."}
    {-expdate string expdate
           "When does the account expire."}
⋮
    {-uid integer uid
         "User Identifier of the principal to be added."}}
#
# Initializing some variables.
#
    upvar 1 local_args cargs
    set local_args $cargs
    set account_args ""
    set princ_args ""
    set group_args ""
    set force 0
    parseargs $arg_table local_args -no_leftovers
    if { [info local help_prose ] > 0 } { return }
    if { [llength $local_args] > 1 } {
      error "Unrecognized argument [lindex $local_args 1]."
    } elseif { [llength $local_args] == 0 } { error "No user name."
    } else { set account_name $local_args }
#
#  If parseargs returned attributes in a list instead of options,
#  create an attribute list. Then call process_attribute_list to
#  parse the list.
#
    if { [info local attribute_list] > 0} {
       set pile_of_attributes "alias client descr dupkey expdate\
       forwadabletkt fullname force group home organization maxtktlife \
       maxtktrenew mypwd password postdatedtkt proxiabletkt pwdvalid \
       renewabletkt server quota shell stdgtauth"
       process_attribute_list attribute_list $pile_of_attributes
    }
#
# If user entered attributes as options rather than in a list,
# check for attribute options.
#
    if { [info local group] > 0} {
      set account_args [format "%s {%s %s}" $account_args group $group]
    } else { error "No group name specified." }
    if { [info local organization] > 0} {
      set account_args [format "%s {%s %s}" $account_args organiz $organization]
    } else { error "No organization name specified." }
    if { [info local password] > 0} {
      set account_args [format "%s {%s %s}" $account_args password $password]
    } else { error "No password specified." }
    if { [info local mypwd] > 0 } {
      set account_args [format "%s {%s %s}" $account_args mypwd $mypwd]
    } else { error "No admin password specified." }
#
# principal and group operations both use the principal's fullname
#
    if { [info local fullname] > 0 } {
      set princ_args [format "%s {%s {%s}}" $princ_args fullname $fullname]
      set group_args [format "%s {%s {%s}}" $group_args fullname $fullname]
    }

    if { [info local uid] > 0 } {
      set princ_args [format "%s {%s %s}" $princ_args uid $uid]
    }
⋮
```

```
    if { [info local stdtgtauth] > 0 } {
       set account_args [format "%s {%s %s}" $account_args stdtgtauth \
                         $stdtgtauth]
    }
#
# set variables if entered as attributes in an attribute list
#
    set account_name [lindex $account_name 0]
    set group_created 0
    set org_created 0
    set group_arg ""
    set org_arg ""
#
# do the work - create principal, do group and organization
# operations, create the account, and create directory in CDS
#
    foreach element $account_name {
      set clup_user "_dcp_cleanup_user_create $element -principal"
      _dcp_create_principal_entry $element $princ_args

      if { $force == 1 } {
      if {[ catch {_dcp_create_group $group group_created} msg] != 0 } {
               _dcp_cleanup_user_create $element -principal
               error $msg
          }
          if { $group_created == 1 } {
             set group_arg "-group group"
          }
          if {[ catch {_dcp_create_org $organization org_created} msg] != 0 } {
              set clup_user [concat $clup_user $group_arg]
              eval $clup_user
              error $msg
          }
          if { $org_created == 1 } {
              set org_arg "-org organization"
          }
      }
      set clup_user [concat $clup_user $group_arg $org_arg]
      if {[catch {_dcp_add_group_entry $group  $element} msg] != 0} {
          eval $clup_user
          error $msg
      }

      if {[catch {_dcp_add_org_entry $organization $element} msg] != 0 } {
           eval $clup_user
          error $msg
      }

      if {[catch {_dcp_add_account_entry $element $account_args} msg] != 0} {
          eval $clup_user
          error $msg
      }

      if {[catch {_dcp_add_namespace_entry $element} msg] != 0} {
          eval $clup_user
          error $msg
      }
    }
    set _n $account_name
    return
}
#
# _dcp_cleanup_user_create  - This function undoes changes after a
```

```
# failure in one of the user create functions as though the operation
# never occurred
#
proc _dcp_cleanup_user_create {account_name args} {

     if { [lsearch $args -principal] != -1 } {
        principal delete $account_name
     }
     if { [lsearch $args -group] != -1 } {
        upvar 1 group clean_group
        group delete $clean_group
     }
     if { [lsearch $args -org] != -1 } {
        upvar 1 organization clean_org
        organization delete $clean_org
     }
}
#
# process_attribute_list - Takes an attribute_list and parses out the
#                          appropriate attributes contained in the
#                          pile_of_attributes variable
#
proc process_attribute_list {attribute_list pile_of_attributes} {
   foreach element $pile_of_attributes { upvar 1 $element _dcp_$element }
   upvar 1 attribute_list _dcp_attribute_list
   set _dcp_attribute_list [check_list_list $_dcp_attribute_list]
   foreach element $_dcp_attribute_list {
       if { [llength $element] != 2 } {
           error "Incorrect attribute list element "$element"."
       }
       set attribute_name [lindex $element 0]
       set attribute_value [lindex $element 1]
       set _dcp_attr_name [info vars _dcp_$attribute_name*]
       if {[llength $_dcp_attr_name] > 1} {
           error
            "Ambiguous attribute "$attribute_name" could be: $_dcp_attr_name."
       }
       set [set _dcp_attr_name] $attribute_value
   }
}
proc check_list_list {attribute_list} {
   set not_list_list 0
   set i 1
   foreach element $attribute_list {
        if {[llength $element] != 2 && [llength $attribute_list] < 3} {
           if {$i == 1} {
              return [format "{%s}" $attribute_list]
           }
        }
        incr i
   }
   return $attribute_list
}
```

The next procedure we discuss in the **user** task object is one that takes a single optional argument and returns lots of output information — **_dcp_show_user**. This procedure returns the results of **principal show**, **principal catalog**, and **account show** operations.

```
#
#_dcp_show_user - This procedure shows the principal and account
#                 attribute lists for a specified user.
#
proc _dcp_show_user {local_args} {
    upvar 1 local_args cargs
    set local_args $cargs

    parseargs "" local_args -no_leftovers
    if { [info local help_prose ] > 0 } { return }
    if { [llength $local_args] > 1 } {
      error "Unrecognized argument [lindex $local_args 1]."
    } elseif { [llength $local_args] == 0 } { error "No user name."
    } else { set account_name $local_args }
    # Take the first element of the account_name in order to
    # eliminate list nesting.
    set account_name [lindex $account_name 0]
    set _dcp_principals [principal catalog -simplename]
    # Show each account that has been requested.
    foreach element $account_name {
      if { [lsearch $_dcp_principals $element] == -1 } {
        error "User \"$element\" does not exist."
      } else {
        set _dcp_user_attributes [principal show $element]
      }
      set _dcp_accounts [account catalog -simplename]
      if { [lsearch $_dcp_accounts $element] == -1 } {
        error "User \"$element\" does not exist."
      } else {
      set _dcp_user_attributes [format "%s\n%s" \
                              $_dcp_user_attributes \
                              [account show $element -all]]
      }
    }
    return $_dcp_user_attributes
}
```

Using the parseargs Procedure

Task objects and scripts that take arguments or options can call the **parseargs** procedure to parse arguments passed along with the object or script invocation. The **parseargs** procedure is a script in a separate file which provides a convenient and reusable method for argument parsing within a **dcecp** script. The basic syntax is:

```
parseargs {parse_options local_args args}
```

The procedure relies on arguments passed to it by the calling script. The **parseargs** procedure requires the following inputs:

parse_options The argument table (**arg_table**) describing the parsing options. The parse_options can consist of five elements as in the script's top level argument table or four elements as in lower-level argument tables for called procedures within a script. The two syntaxes for parse_options are:

verb **command** *variable command_name* **"***help string***"**
-*options type variable* **"***help string***"**

verb Provides top-level parsing. Typically an operation contains an object and a verb. The verb portion generally calls another procedure.

command A keyword indicating that the procedure being defined is a verb of an object.

variable The name of the variable which hold the value of the option. When parsing verbs, the variable is named *function call*. When parsing options, the variable is named for the option being parsed. For example, if the option name is **–alias**, the variable is named **alias**.

command_name
The procedure name to store in the variable

help string The string which describes the use of the verb or option.

–options The actual string value of the option to be parsed such as **–attribute** or **–mypwd**.

type The type of variable to be associated with **–option**. Acceptable types are: **integer**, **string**, **float**, **boolean**, **command**, and **help**.

local_args The arguments to be parsed. The parseargs procedure extracts all of the recognized entries into a list and resets *local_args* with the values that were not parsed (or not parsable). For instance, a top level command like **user create** includes options that are parsed later when the procedure implementing the **create** operation is invoked within the script.

args One of two flags:

–found_one
Tells the parser to return when one procedure argument has been found. So, in **user create**, the parser would return after create command had been found and processed.

–no_leftovers
Looks for extra options and generates an error if one is found.

Steps — Formal Task Objects

After your task object is written (and tested), you need to make it available for use. If your script is intended just for your personal use, you can include it in your **.dcecprc** file and invoke it as described in “Informal Administration Scripts” on page 2-39.

Formal task objects require a few steps to make them behave like other **dcecp** objects.

1. Log in as root and copy the finished script into the /**opt**/**dcelocal**/**dcecp** directory and set the file permissions to executable.
2. Start the **dcecp** program and run the **auto_mkindex** utility. This creates information that informs the DCE control program about all available objects. With root privileges, run the following command in the directory where the task objects reside. On UNIX systems, this is often the /**opt**/**dcelocal**/**dcecp** directory.

   ```
   % dcecp
   dcecp> auto_mkindex /opt/dcelocal/dcecp *.dcp
   dcecp>
   ```

3. To include the new task object name in the **dcecp** help screen, edit the file /**opt**/**dcelocal**/**dcecp**/**help.dcp**. This file is displayed in response to the **dcecp help** operation.

   You need to make this file available on each DCE host where the script will be executed. Generally this means copying the file to each host's /**opt**/**dcelocal**/**dcecp** directory and then running the **auto_mkindex** utility on the files in the directory. You might want to place the object name in the /**opt**/**dcelocal**/**dcecp**/**help.dcp** file as well.

   As a convenience, you could write a script that uses the DCE control program's **hostdata** object to create the file on each host. The script could then run **auto_mkindex** utility using the **hostdata**

object's postprocessor attribute. “Managing DCE Host Services and Host Data” on page 1-28 contains information on using the **dcecp hostdata** object.

Grouping Elements and Controlling Interpretation

Introduction — Grouping Elements

Programming languages often use braces, quotes, parentheses, and so on to operate on selected elements as a group rather than individually. Similarly, **dcecp** uses double-quotes and braces to group elements into structures called lists. Lists are a special way to group elements so that **dcecp** can correctly parse commands and other data like return values.

More Detailed Information — Grouping Elements

The **dcecp** command elements are separated by white space — the space, tab, and new line characters. The following **dcecp** command uses space characters to separate its three elements:

```
dcecp> directory create /.:/subsys/comm_services
dcecp>
```

The choice to use braces or quotes to group elements together depends on how you want **dcecp** to interpret special characters like $, [, and {. While braces disable special interpretation of most of these characters, double quotes disable special interpretation of just a few. The backslash character (\) which we will see soon, offers another way to disable interpretation of special characters. When used together, braces, quotes, and backslashes offer lots of flexibility in composing **dcecp** command strings.

Using Braces to Group Elements into Lists

Braces group separate elements together creating a new element that consists of everything between a left brace and its corresponding right brace. You can also nest braced elements.

Braces disable command [], variable $, and backslash (\) substitution. While the most important use of braces is to ensure a **dcecp** command has the correct number of arguments, this also provides a convenient way to include special characters in a list.

While the **puts** command is often used for writing to files, when called with only one argument it writes the argument to **stdout**. In our example, the first use of **puts** allows normal interpretation of the variable *a*. The second use of **puts** groups the separate elements into one argument by disabling special interpretation of space characters and the dollar sign.

Using Double Quotes Instead of Braces

Like braces, double quotes also group elements together. But unlike braces, double quotes cannot be nested. Furthermore, while braces disable almost all special characters, double quotes disable just a few — spaces, tabs, new lines and semicolons— letting you avoid the potentially awkward use of backslashes in a string of text elements. The most convenient use of double quotes is to allow clean, readable expansion of variables using the dollar sign ($) trigger.

Use of double quotes does not disable command, variable, and backslash substitution.

Using Backslashes to Include Special Characters

We already know that **dcecp** relies on certain special characters such as spaces, braces, quotes, or dollar signs to control its interpretation of elements. Sometimes, you might want to include one special character

in a string, temporarily suspending its special interpretation. The backslash (\) character provides a form of substitution that suppresses special interpretation of the character immediately following the backslash.

Use the backslash character (\) to insert a nonprinting space character in a string of elements.

The following list shows the special characters that you can include in a string of elements using the backslash character.

\b	Backspace
\t	Tab
\e	Escape
\n	Newline
\r	Carriage-return
\{	Left brace
\}	Right brace
\[	Open bracket
\]	Close bracket
\$	Dollar sign
\ (space)	Space " "
\;	Semi-colon
\"	Double quote
\\	Backslash
\newline	Nothing
\ddd	Octal value

Examples — Grouping Elements

- Use either the new line character or the semicolon (;) to separate commands in a script. The following two examples which set and then use a variable are equivalent:

```
dcecp> set a /.:/subsys/comm_services
/.:/subsys/comm_services
dcecp> directory create $a
dcecp>

dcecp> set a /.:/subsys/comm_services; directory create $a
dcecp>
```

- Each of the following example lists contain three elements:

```
larry moe curly

1 {3 5 7 11 13} {17 19}

red {orange yellow {green blue} indigo} violet
```

- Consider the use of braces to include special characters in a list in the following example:

```
dcecp> set a solution
solution
dcecp> puts $a
solution
dcecp> puts {This is a convenient $a}
This is a convenient $a
```

- In the following example we set a variable (**a**) to a value that includes spaces.

```
dcecp> set a "XYZ server for /.:/corp/comm_groups"
XYZ server for /.:/corp/comm_groups
dcecp> puts $a
XYZ server for /.:/corp/comm_groups
dcecp>
```

- The following is a variation of the previous braces example:

```
dcecp> set a solution
solution
dcecp> puts $a
solution
dcecp> puts "This is a convenient $a."
This is a convenient solution.
dcecp>
```

  In this example, the use of quotes with the second **puts** command gathers five elements into a single argument for **puts** by disabling special interpretation of the space characters. However, the quotes do not affect interpretation of the dollar sign ($) character.

- Each of the following **dcecp** lists have three elements:

```
a b\ c d

a b \{
```

  The elements in the first example are a, b c, and d. The elements in the second example are a, b, and {. A more practical example could use the backslash to include quotes in error messages as shown in the following code fragment:

```
if {[llength $a] < 2} {
   error "Unable to parse \"$element_list\"."
}
```

Informal Administration Scripts

Introduction — Informal Administration Scripts

Informal administration scripts let administrators store multiple operations in a file and replay them whenever necessary. Informal scripts are useful for operations that take only one or two arguments or that just perform simple tasks. Furthermore, the script's precise behavior and output can be custom-tailored to the needs of whoever is writing it. While informal scripts can be shared among administrators in a cell, they are typically included just in the author's **.dcecprc** file.

More Detailed Information — Informal Administration Scripts

Scripts generally consist of one or more procedures created with the **proc** command. This lets you invoke the scripted operation by simply typing the procedure's name at the **dcecp** prompt. The following simple script prints information about who you are in terms of your current cell and login identity.

```
# Show your current login name and your current cell name.
proc _dcp_whoami {} {
  global _c _u
  puts stdout "You are '$_u' logged into '$_c'."
}
```

This script can be included in your **.dcecprc** file either directly or by using the **source** command and keeping the actual script in an external file. The second method lets other administrators include your same script by simply pointing to it with **source** commands in their **.dcecprc** files. This method also keeps your **.dcecprc** file uncluttered making it easier for others to understand what is going on. Alternatively, you can place the script or a pointer in the **init.dcecp** file. changes to this file are available to all users on a host. For more information about the **init.dcecp** file and the **.dcecprc** file, see “Customizing **dcecp** Sessions” on page 2-13.

You can create scripts that do more by chaining operations together. For example the following script lists all the hosts in a DCE cell. Then it checks whether each host has an object entry in CDS for a dts-entity (this would indicate that a DTS server is available on the host). For each host with an object entry for a dts-entity, the script does a **clock show** operation which returns the time on that host. The script prints the information on the display formatting it for readability and continues looping through all the hosts in the cell until all host entries have been checked.

Examples — Informal Administration Scripts

- An example of the **source** command in a **.dcecprc** file is:

  ```
  source /usr/users/wardr/dcecp/local_lib.dcp
  ```

  The **.dcp** file name extension is a convention for naming files used by the DCE control program. Another convention precedes procedure names with **_dcp** as in **_dcp_whoami**. Many **dcecp** procedures adhere to this convention to distinguish their names from user-created procedures which do not need to use this convention. If you find procedure names like **_dcp_whoami** hard to remember or type, you can rename them.

- You might rename the sample procedure to **whoami** using the **rename** command in the **.dcecprc** file:

  ```
  rename _dcp_whoami whoami
  ```

  Restart the **dcecp** program to pick up any changes. Now you can type `whoami` at the DCE control program prompt:

  ```
  dcecp> whoami
  You are 'cell_admin' logged into '/.../my_cell.goodco.com'.
  dcecp>
  ```

- Make the **_dcp_show_clocks** procedure available to your **dcecp** session in the same way as the simpler script described previously.

  ```
  # Show the time on all of the dts servers running in your cell.
  proc _dcp_show_clocks {} {
      set x [directory list /.:/hosts]
      foreach n $x {
          if {[catch {object show $n/dts-entity}] == 0} {
              set index [string last "/" $n]
              set y [string range $n [incr index] end]
              if {[catch {clock show $n/dts-entity} msg] == 0} {
                  set i [expr 20 - [string length $y]]
                  puts [format "Time on $y is %${i}s %s" " " \
                      [clock show $n/dts-entity]]
              } else {
                  set i [expr 20 - [string length $y]]
                  puts [format "Time on $y is %${i}s %s" " " \
                      "Server not responding."]
              }
          }
      }
  }
  ```

Interactive Command or Script Usage

Introduction — Interactive Command or Script Usage

Note: Entering the following sample commands may generate an error if you have not created the account referenced in these examples. Replace the variable information with information that is valid for your system.

There's no absolute dividing line for when you should enter commands interactively or with a script. In general though, the simpler operations — those that perform one or maybe two tasks — make the best candidates for interactive use.

More Detailed Information — Interactive Command or Script Usage

Saving a frequently used operation as a script (in a file) has its advantages; it can help to automate repetitive or complicated tasks and you can keep it around for possible modification and use in other situations later on. Whichever method you choose, as you become more comfortable using **dcecp** and Tcl, you might find yourself entering fairly complex operations interactively. For information on how to create and invoke scripts, refer to “Customizing **dcecp** Sessions” on page 2-13.

Examples — Interactive Command or Script Usage

- The following examples typify interactive operations:

```
dcecp> directory create /.:/printers

dcecp> account show w_shakespeare

dcecp> server start /.:/hosts/curley/config/srvrconf/BBSserver
```

  Note: Entering these sample commands may generate an error if you have not created the accounts referenced in these examples. Replace the variable information with information that is valid for your system.

- The next example is a little more complicated so at first, you might choose to run this as a script.

```
foreach i [group list temps] {
   account modify $i -expdate 1996-06-30}
```

Looking Beyond the Tools

Although you use the task objects to perform various administrative operations, your most important focus is on the elements or entities that you're managing. Each of four task objects provided with DCE enables you to manage a specific element or entity in your DCE cell. The elements are as follows:

A DCE Cell You can test whether a cell is running, show general information about available services in a cell, and back up security and CDS information by using the **cell** task object.

DCE Hosts You can configure and remove DCE hosts in a cell, show information about hosts in a cell, and start and stop DCE processes on hosts in a cell by using the **host** task object.

DCE Users You can add and remove users and show information about users in a DCE cell with the **user** task object.

Manipulating Strings

Introduction — Manipulating Strings

Many DCE administrative operations return information of some sort. For instance, the **principal show** operation returns information about a principal. Usually this information is in the form of a list as in the following example:

```
dcecp> principal show R_Parsons
{fullname {}}
{uid 15}
{uuid 0000000f-d6f9-21cd-8d00-0000c08adf56}
{alias no}
{quota unlimited}
{groups users}
dcecp>
```

More Detailed Information — Manipulating Strings

Although it is fairly easy for an administrator to scan a list and extract the necessary information from it, scripts operate in the dark, feeling their way through information. When scripts search for specific information, they usually ignore the notion of lists, operating instead on the collection of characters (called a *string*) that makes up a list. The DCE control program provides a set of commands to operate on strings, letting you construct, parse, compare, extract values from, and modify strings.

Constructing Strings

Often, scripts need to construct strings for use in other commands or for displaying on the screen for users. The DCE control program provides a **format** command that you use to construct strings for use by your script.

The **format** command substitutes variables where needed.

The **format** command can also convert arguments between differing forms including decimal, octal, hexadecimal, floating point, and scientific notation. You can also specify to print or omit signs for signed numbers, right- or left-justify output, and padding with spaces or zeros.

Parsing Strings

The DCE control program includes a **scan** command that parses strings and then converts, and stores relevant parts of strings in variables. This capability is useful, for instance, when converting information returned by a previous command into data that can be input to another command. The syntax for the **scan** command is:

```
scan "string"  "format"  [varname [varname]...]
```

You can specify the *string* literally or by using a variable. The *format* section controls parsing, ignoring blanks and tab characters you might have included in the *format* section for readability. This section consists of one or more conversion specifiers delimited by percent signs (%). Conversion specifiers define which parts of *string* get converted and stored, as well as the type of conversion.

Other String Handling Operations

You can specify one character or a range of characters in a string using **string index** and **string range**. These commands would be useful for extracting information from a string of predictable length.

The **string index** command has one argument that is the position of one character (counting from left to right beginning with character zero) to be extracted from the string. The **string range** command includes two arguments that are the positions of the leftmost and rightmost characters to be included in the range.

You can determine whether one string is lexicographically (alphabetically) greater than, less than, or equal to another string using **string compare**. Generally, this operation performs a byte comparison of ASCII codes that make up the string.

Trim specific characters from a string using the **string trim** command. Remove the leftmost or rightmost characters from a string using the **string trimleft** and **string trimright** commands.

You can perform pattern matching operations in any of several ways. Invoke *glob* style pattern matching with the **string match** command. This mimics the glob pattern matching capabilities available in **csh**, returning a 1 for a match and a 0 for no match. More flexible regular expression pattern matching (like that found in **egrep**) can be performed using **regexp** command. You can extend this operation to perform regular expression substitution using the **regsub** command.

Examples — Manipulating Strings

- The following example constructs the variable *_dcp_host_entries* using the **format** command to prepend the cell name string (the **string** type is indicated by **%s**) to the string /**hosts**. The cell name is contained in the **_c** convenience variable.

```
dcecp> set _dcp_host_entries [format "%s/hosts" $_c]
/.../my_cell.goodco.com/hosts
dcecp>
```

- The following examples convert the integer 8 to its octal equivalent. The second example shifts the output nine character spaces to the right.

```
dcecp> format %1o 8
10
dcecp> format %9o 8
       10
dcecp>
```

- The following example parses the string contained in the variable *_dcp_temp* for a valid floating point number and stores it in the variable *_dcp_temp2*.

```
if { [scan $_dcp_temp "%f" _dcp_temp2] != 1 } {
   error "Variable \"$_dcp_temp\" is not a valid floating point number"
}
```

- The following example illustrates one use of the **string range** command.

```
dcecp> string range {The quick brown fox} 4 9
quick
dcecp>
```

- Count the number of characters in a string using the **string length** command. For instance:

```
dcecp> string length "The quick brown fox"
19
dcecp>
```

- Convert characters between upper and lower case using the **string toupper** and **string tolower** commands. For example:

```
dcecp> string toupper "The quick brown fox"
THE QUICK BROWN FOX
dcecp>
```

- The following example illustrates the use of the **regsub** command. The first argument specifies the search pattern. The second argument is the string to search. The third argument specifies the replacement pattern. The last argument is a variable into which **regsub** places the new string. The command returns a 0 if no substitution occurs and a 1 if substitution does occur.

```
dcecp> regsub brown "The quick brown fox" blue color
1
dcecp> puts $color
The quick blue fox
```

Measuring and Counting with Expressions

Introduction — Measuring and Counting with Expressions

The **expr** command offers flexible ways to express and use arithmetic functions in your scripts. Expressions are useful for things like comparing numeric information such as the number of elements in a list, setting thresholds for monitoring purposes, incrementing counters that control your script's execution, and producing statistical information.

More Detailed Information — Measuring and Counting with Expressions

A simple **dcecp** expression is a combination of an operator like add (+) or multiply (*) and some operands. The **expr** command takes one argument—the expression—so parenthesis or braces may be needed if your expression has spaces. Use parenthesis to control grouping in expressions. Expressions can also be nested. All of the following are valid expressions:

```
dcecp> expr {2 + 3}
5
dcecp> expr 2+3
5
dcecp> set x 24
24
dcecp> expr ($x-8)*2
32
dcecp> expr $x-(8*2)
8
dcecp> expr $x-8*2
8
dcecp>
```

Be careful using variables in expressions; variables like *$x* above must be numeric strings like 24, not non-numeric strings like 4*6.

The **dcecp** program normally treats numbers as integers in decimal format, but you can read numbers in octal and hexadecimal formats too. Precede a number with 0 for octal interpretation as in 0477. Precede a number with 0x for hexadecimal interpretation as in 0x9FF. You can also represent numbers in floating point using any of the forms specified by the ANSI C standard (with the exception of the f, F, l, and L suffixes).

The **dcecp** program also supports numerous mathematical functions in expressions such as cos, exp, log, tan, sin, and others, by invoking the C math library functions of the same name.

Here is a partial list of operators you can use with the **expr** command. The list order also denotes precedence. This means, for instance, that **expr** multiplies before adding (2+2*4 equals 10).

–	unary minus
~	bit-wise NOT
!	logical NOT
*	multiply
/	divide
%	remainder
+	add
–	subtract
<<	left shift
>>	right shift
<	boolean less than
<=	boolean less than or equal
>	boolean greater than
>=	boolean greater than or equal
==	boolean equal
!=	not equal
&	bitwise AND
^	bit-wise exclusive OR
|	bit-wise OR
&&	logical AND
||	logical OR
a?b:c	if-then-else (as in C).

Operating on Lists

Introduction — Operating on Lists

Lists provide convenient ways to operate on collections of things such as sets of principals, group members, or other objects. Lists are collections of objects entered by you or returned from commands. They are any number of elements separated by spaces, tabs, or new lines. Usually, a list is enclosed in braces. Both of the following are examples of lists.

```
{n_long l_jones p_sawyer d_witt m_dougherty s_preska}
```

```
{{/.:/hosts} {/.:/subsys}}
```

More Detailed Information — Operating on Lists

The **dcecp** program relies on lists to group elements so they can be correctly parsed by the **dcecp** command interpreter. For example, the set command takes two arguments:

```
set varName value
```

The following **set** command cannot be correctly parsed because **dcecp** detects a third argument.

```
dcecp> set a John Hunter
Error: wrong # args: should be "set varName ?newValue?"
dcecp>
```

Use braces, quotes, or backslashes to create a valid list.

```
dcecp> set a {John Hunter}
John Hunter
dcecp> set a "John Hunter"
John Hunter
dcecp> set a John\ Hunter
John Hunter
dcecp>
```

The commands that operate on lists provide convenient ways to evaluate, select, and act on individual elements or groups of elements in a list. The **dcecp** program provides a comprehensive set of commands that let you create, modify, search, sort, and convert to and from lists.

The **dcecp** program provides numerous commands for working with lists. You can join lists together using the **concat** command. Use **linsert** to add elements to an existing list. Extract a range of elements using **lrange**, replace elements in a list with **lreplace**, and sort list elements in alphabetical (dictionary) order using **lsort**. The **dcecp** program also includes an **attrlist** object (see the **attrlist** reference page in the *DCE for AIX Administration Command Reference* for use in manipulating list elements).

Examples — Operating on Lists

- The following script returns the last element in a list. The **llength** command returns the number of elements in the list. Our list has four elements so **llength** returns 4. The **dcecp** program numbers the elements from left to right starting with 0 so our list with three elements has elements numbered 0, 1, 2, and 3. The value of variable *c* is set to the number of the last element in the list (3). Finally the **lindex** command returns element 2 (*f*).

  ```
  dcecp> set a {a b {c d e} f}
  a b {c d e} f
  dcecp> set b [llength $a]
  4
  dcecp> set c [expr $b-1]
  3
  dcecp> lindex $a $c
  f
  dcecp>
  ```

- Here is an example that lists all child directories in a tree in alphabetic order. **_r** is a **dcecp** convenience variable that holds the output of the last command. In this case, **_r** holds the list of directories returned by the **directory list –simple** command.

  ```
  dcecp> directory list -simple /.:
  hosts subsys cell-profile fs lan-profile planets_ch sec sec-v1
  dcecp> lsort $_r
  cell-profile fs hosts lan-profile planets_ch sec sec-v1 subsys
  dcecp>
  ```

Reading Other Files as dcecp Scripts with source Command

Introduction — Reading Other Files

The **source** command reads the contents of other files, executing them as **dcecp** scripts. This capability lets you construct higher level scripts by plugging lower level functions together—like building blocks. Because you reuse your scripts rather than duplicate them with potential variations, scripts are more consistent and easy to develop and maintain. The command syntax is:

```
dcecp> source filename
dcecp>
```

The return value from **source** is the return value from the last command in *filename*.

More Detailed Information — Reading Other Files

Terminate a **source** command using the **return** command. The **return** command provides a way for commands like **source** and **proc** to exit in a controlled manner, even when expected or unexpected error conditions occur. Rather than allow error conditions to cause the whole script to exit and fail, the **return** command manages error information and allows the script to continue executing. We discuss the use of **return** with other error-handling techniques in “Manipulating Strings” on page 2-41.

Example — Reading Other Files

As a practical example, imagine we have one script that lists entries in CDS subtrees, another script that deletes subtrees, and another script that moves subtrees. One common function needed by all these scripts might be to list every child directory under the root of the subtree. You could write a script that lists every child and name it something like **children_list.dcp**. (The **.dcp** extension is a **dcecp** convention for naming script files.). When any of your scripts need to list all the child directories, simply use the **source** command:

```
source children_list.dcp
```

Spawning Subprocesses

Introduction — Spawning Subprocesses

Using subprocesses to execute commands offers several convenient solutions to some complex scripting or special administrative needs. Subprocesses can provide:

- Access to operating system commands;
- A way to establish synchronous, orderly execution; and
- Ways to streamline complex or sophisticated scripts.

More Detailed Information — Spawning Subprocesses

Although the DCE control program is versatile, there are times when you may want your script to use operating system commands to accomplish some simple (or even not-so-simple) operation. The **exec** command provides a way for scripts to perform external commands by forking a subprocess in which the command executes.

The **exec** command normally returns the results of the operation performed in the subprocess. However, you can use AIX-style redirection symbols (<, <<, and >) to redirect standard input or standard output. You can also use the vertical bar (|) symbol to pipe the output through filters such as **nroff**, **sort**, or **grep**.

When used alone, the **exec** command is synchronous, meaning that the external command completes before the script continues executing. But when a subprocess will take a long time to complete, for instance when you synchronize directories in a CDS cell, you can use the **exec** command with an ampersand (&) to push a subprocess into the background.

Examples — Spawning Subprocesses

- The following example uses the **exec** command to retrieve the local host name which is then established as a *hostname* variable and subsequently used in the script.

```
dcecp> set hostname [exec hostname]
myhost
dcecp> directory list /.:/hosts/$hostname -simple
cds-clerk cds-server dts-entity profile self
dcecp>
```

- The following example uses the exec command to send previously collected output to a printer. This lets your script continue without having to wait for the print command to complete.

```
dcecp> exec lpr output.log &
dcecp>
```

Variable Substitution

Introduction — Variable Substitution

Like other programming languages, **dcecp** provides shorthand ways to express and use values. Variable substitution is one shorthand method that lets you represent a value—say, the name of an object in a CDS directory — as a variable.

Use the Tcl **set** command to establish a value for a variable. For readability, a variable name can consist of any combination of letters, numbers, and underscore characters. Use quotes or backslashes (\) to include spaces in variable names (although this is not usually recommended) or values.

Examples — Variable Substitution

- All of the following lines use valid variable names:

```
set a $i
set CDS_clearinghouse_name cambridge_ch
set DCE_user_1 "William Rosenberry"
```

- The following example sets variable *a* to have a value of 7. The second use of the **set a** command without a value causes **dcecp** to display the current value of the variable.

```
dcecp> set a 7
7
dcecp> set a
7
```

- After establishing a value for a variable using the Tcl **set** command, you can use the variable elsewhere in your script or interactive command. The **dcecp** program uses the dollar sign to trigger insertion of the current value into the command word. A simple example is:

```
dcecp> set a 7
7
dcecp> expr $a+2
9
```

- In the previous example we first set variable a to 7. In line 2, we use the **expr** command to add 2 to the value of **a** (7). The dollar sign triggers **dcecp** to insert the value 7. The last line shows the return value from the **expr** command.

  A more relevant example might be:

```
dcecp> set a /.:/sec
/.:/sec
dcecp> object show $a
{RPC_ClassVersion
 {01 00}}
{RPC_ObjectUUIDs
 {06 3b 23 00 72 e5 e0 1d 8c b4 00 00 c0 8a df 56}}
{RPC_Group
 {2f 2e 2e 2e 2f 77 61 72 64 5f 63 65 6c 6c 2e 6f 73 66 2e 6f 72
 67 2f 73 75 62 73 79 73 2f 64 63 65 2f 73 65 63 2f 6d 61 73 74
 65 72 00}}
{CDS_CTS 1994-05-23-17:21:37.481+00:00I0.000/00-00-c0-8a-df-56}
{CDS_UTS 1994-05-23-17:22:36.607+00:00I0.000/00-00-c0-8a-df-56}
{CDS_Class RPC_Group}
{CDS_ClassVersion 1.0}
dcecp>
```

- Remove (undefine) a variable using the Tcl **unset** command as in the following example:

```
dcecp> unset a
dcecp> set a
Error: can't read "a": no such variable
dcecp>
```

Working with Files

Introduction — Working with Files

The DCE control program has several commands for use in reading from and writing to files. Files are useful for things like storing the output of dcecp operations for later reference. Here are several useful examples of file manipulation.

- You could run a **server catalog** operation across all of the hosts in a cell and store the results from each host in a host-specific file. Later, you could compare the files to produce a report of server configurations.
- You could detect inactive accounts by running a **dcecp** script that shows the last time each account was logged into, storing this information in a file for later evaluation.
- You could also modify DCE files that are not manipulated easily using the **dcecp hostdata** object. For example, you could write a function that added a new attribute to the **cds_attributes** file.

DCE as provided by the Open Software Foundation currently supports file operations only for UNIX-style systems or for systems that support POSIX system calls. However, some vendor DCE versions may support file operations on non-UNIX systems.

More Detailed Information — Working with Files

Specifying File Names

Specify file names using customary UNIX-style names. For instance **/opt/dcelocal/dcecp/server_snap.dcecp** refers to a file named **server_snap.dcecp** in a directory called **/opt/dcelocal/dcecp**. You can also refer to files using relative file names like **~dce_admin/scripts/server_snap.dcecp** and **~/admin/server_snap.dcecp**. You can print the current working directory using the **pwd** command and set the current working directory using the **cd** command. The following command sets the current directory to be **~dce_admin/scripts**.

```
dcecp> cd ~dce_admin/scripts
dcecp>
```

You can view a list of files in a directory using the **glob** command. This command returns a list of file names that match pattern arguments to the command. For instance:

```
dcecp> glob *
help local_lib.dcp
dcecp>
```

You can view lots of other information about files using the **file** command with various options. The **file** commands can help select a file based on its age, its size, or its permissions (whether it is executable, or readable, or writable by the current user).

Reading and Writing Files

The **dcecp** commands for reading and writing to files look and act like their C language counterparts — **fopen**, **fclose**, and so on.

Open a file for reading and writing using the **open** command. The second argument to the **open** command (shown in the following example as **r+**) specifies the file access mode. You can open files for reading, or writing, or both and you can specify whether to replace existing files or to add to them with new information.

The **open** command assigns a file identifier to each file when it is opened. Use the file identifier to refer to files in subsequent commands.

When a file is open, you can add lines to a file using the **puts** command. Normally, **dcecp** waits until it has accumulated sufficient data before writing this information to a file. If you want **dcecp** to immediately write the information to a file, use the **flush** command. Use **gets** to read the next line from a file or use **read** to read a number of bytes or all of the bytes in a file.

Sometimes, you do not want to start reading or writing at the first line of a file. The DCE control program provides several commands that set the access position so you do not have to advance through every line in the file. These commands will produce an error if you use them for devices like terminals or other sequential devices that do not support random access. Use the **seek** command to set the access point in a file. Specify the offset as a number of bytes from the origin which can be the beginning or end of the file or the current position.

You can determine the current access position using the **tell** command. Save the return value in a variable so you can go back to that position in the file later on.

Examples — Working with Files

- You can set the initial access position to the beginning or the end of a file. The default access mode is read only (the file must already exist).

  ```
  dcecp> open server_snap.dcecp r+
  file5
  dcecp>
  ```

- The following example writes a list of all principals in a file named **prins**:

  ```
  dcecp> open prins w+
  file8
  dcecp> puts file8 [principal catalog]
  dcecp> close file8
  dcecp>
  ```

- Use a negative number to move toward the beginning of the file as in the following example which moves back sixteen bytes from the current access position.

```
dcecp> seek file5 -16 current
dcecp>
```

- Finally, you can close a file using the **close** command.

```
dcecp> close file5
dcecp>
```

Chapter 3. DCE Administration Tasks

This section describes administrative tasks used to manage cells, hosts, and users. For introductory and overview information on these administrative tasks, see “DCE Administrative Tasks Overview” on page 1-26. That section also groups related tasks together in the order in which they may normally occur so you can access these referenced tasks in a logical, tutorial-type manner. You may find this helpful in performing related tasks together from beginning to end.

Backing Up the Security Service Registry and CDS

Introduction — Backing Up Security Service Registry and CDS

As organizations increasingly depend on DCE cells for their day-to-day operations, they cannot afford to lose the cell's directory and security data. Organizations generally rely on regular backup schemes to prevent the loss of this and other critical data. But backing up these DCE databases by using traditional backup methods can cause security holes in your cell if the archives are not properly protected.

Fortunately, DCE includes features that let you back up your databases to destinations of your choosing. After you have begun using the DCE mechanism to back up CDS and security data, you can redirect your traditional backup program to ignore these DCE databases.

Note: The **dced** databases stored in /**var**/**dce**/**dced** are the exception to the above statement. These databases are not backed up when the **cell backup** process is used. You must use your own backup procedure to protect these files.

The **cell backup** operation backs up the master security database and each clearinghouse with master replicas in the cell. This operation requires that a **dced** is running on each of the server hosts being backed up.

For alternate methods of backing up information, see “Backing Up Namespace Information” on page 5-5 and “Backing Up and Restoring the Registry Database” on page 7-36. Also see *DSS for AIX Up and Running!* for general backup information. The *DCE for AIX DFS Administration Guide and Reference* provides information on backing up your DFS databases.

Steps — Backing Up Security Service Registry and CDS

Prepare a cell for regular backup operations by setting up an Extended Registry Attribute (ERA) that can specify a backup destination (typically a tape archive). Then add the new attribute to the principals for the master DCE Security Service registry database and all CDS clearinghouses with master replicas that you want to back up. To do this, follow these steps:

1. Create an extended registry attribute (ERA) as a string that specifies a backup destination. Name the ERA /**.:/sec/xattrschema/bckp_dest** and the type **printstring**. Select the ACL manager named **principal** and set its four permission bits to **r** (read), **m** (manage), **r** (read), and **D** (Delete) as shown in the following command:

   ```
   dcecp> xattrschema create /.:/sec/xattrschema/bckp_dest
                             -encoding printstring -aclmgr {principal r m r D}
   ```

2. Add the new ERA (**bckp_dest**) to the principal **dce-rgy** (the DCE Security Service registry database). Set the value to be the **tar** file name or the device that is the backup destination:

   ```
   dcecp> principal modify dce-rgy -add {bckp_dest tarfilename_or_device}
   dcecp>
   ```

3. Add the new ERA (**bckp_dest**) to the principal /**.:/hosts/***hostname*/**cds-server** (the CDS server). Set the value to be the **tar** file name or the device that is the backup destination:

```
dcecp> principal modify /.:/hosts/hostname/cds-server
                        -add {bckp_dest tarfilename_or_device}
dcecp>
```

Now, whenever you want to back up your registry database or CDS database, just invoke a **cell backup** operation as follows:

```
dcecp> cell backup
dcecp>
```

You can back up another cell by including the cell name as an argument to the **cell backup** operation. Note that you need the necessary permissions in the remote cell. (Refer to the **cell** object reference page for the required privileges.)

Changing the IP Address

Use the following procedure to change the IP address of a machine configured as a CDS and a Security server.

1. Remove knowledge of the clearinghouse on the machine. It is reinstated after the IP address is changed. If you do not know the name, use the **cdscp show cell /.:** command to get it.

   ```
   cdscp clear clearinghouse /.:/<host_ch>
   ```

2. Stop all DCE daemons on the machine:

   ```
   /etc/dce.clean
   ```

3. Remove the endpoint database. It is recreated on restart:

   ```
   rm /opt/dcelocal/var/dced/Ep.db
   ```

4. Remove the clerk cache. It is recreated on restart:

   ```
   cd /opt/dcelocal/var/adm/directory/cds
   rm cds_cache.* cdsclerk_*
   ```

5. Edit the **/opt/dcelocal/etc/security/pe_site** file to reflect the new address so that security can start.

6. Remove old credentials:

   ```
   cd/opt/dcelocal/var/security/creds
   rm *
   ```

7. Change the IP address on your system and reboot. These instructions assume DCE is started automatically using **inittab**. The **gdad** and **dtsd** daemons do not come up since CDS is not completely functional yet. Start these daemons after the conversion process is completed.

8. Since CDS is not available, use the **BIND_PE_SITE** environment variable for security.

   ```
   export BIND_PE_SITE=1
   dce_login cell_admin
   ```

9. Identify to CDS the clearinghouse it is to manage (ensure that you use the same name as in the previous **clear clearinghouse** command):

   ```
   cdscp create clearinghouse /.:/<host_ch>
   ```

10. Because the CDS server was not aware of its clearinghouse when it was started, the **cdsadv** process is also unaware of the existence of this clearinghouse. Rebuild the clerk cache:

```
/etc/dce.clean cds
cd /opt/dcelocal/var/adm/directory/cds
rm cds_cache.* cdsclerk_*
/etc/rc.dce cds
```

The DCS and Security servers are now reconfigured to use the new IP address:

```
unset BIND_PE_SITE
dce_login cell_admin
```

11. Verify that you can now successfully access the namespace:

```
    cdsli -o
/.:/cell-profile
/.:/fs
/.:/lan-profile
/.:/sec
```

12. Update the server self entry in CSD:

```
rpccp unexport -i elaf8308-5dlf-11c9-91a4-08002b14a0fa,3.0
```

13. Change the **/opt/dcelocal/etc/security/pe_site** file so that **sec_clientd** can find the security server on restart.

14. Remove the security credentials:

```
cd /opt/dcelocal/var/security/creds
rm *
```

15. Start the DCE daemons:

```
/etc/re.dce
```

16. Since DCS access is not restored yet, set the **BIND_PE_SITE** variable:

```
export BIND_PE_SITE=1
dce_login cell_admin
```

17. Inform the **cdsadv** process of the new IP address for the CDS server:

```
cdscp define cached server <server_name> tower
```

Configuring a DCE Host in a Cell

After the DCE for AIX 2.1 software is installed on a host, you can configure the local host as a DCE client machine by using the DCE configuration operation. You must have administrator privileges on the local host to execute an operation cell.

You can read more about requirements for DCE server and client systems in “Client and Server Considerations” on page 1-12.

Before configuring a DCE client system be sure the DCE software has been installed on the host. For information about installing and configuring DCE, refer to your DCE installation instructions or the *DSS for AIX Up and Running!*.

Creating a New User

Introduction — Creating a New User

Each **user** in a DCE environment is a person with a unique identity (principal name). Each principal is a member of at least one security group and organization and has an account in the DCE Security Service registry database. Although it is not required, each principal can also have a directory in CDS.

Steps — Creating a New User

When you create a user with the **user** task object, you perform several lower-level operations:

1. The **user create** operation creates a new principal name and adds the principal to a security group and organization. If the security group or organization does not exist when you invoke the operation, you can force their creation using the **–force** option. The principal attributes assume default values, but you can specify other attributes if necessary. All of the attributes are listed in the **user** task object reference page.

   Typically, a security group's name is included in ACLs (Access Control Lists) that regulate user access to various server and data objects in the DCE environment. A security organization maintains policies that are applied to all the principals that are members of that organization. Policies control things like the lifespan of accounts, whether or when account passwords expire, or whether passwords can contain nonalphanumeric characters. You can read more about administering principals, groups, and organizations in “Creating and Maintaining Principals, Groups, and Organizations” on page 1-48.

2. The **user create** operation creates an account for the principal and creates the user's password. The account attributes assume default values but you can specify other attributes if necessary. All of the attributes are listed in the **user** task object reference page.

   A principal's account contains information about the principal such as group and organization names, account creation and expiration information, and information about tickets (which identify principals to resources in a DCE environment). You can read more about administering accounts in “Creating and Maintaining Accounts” on page 1-48.

3. Finally, the **user create** operation adds a directory called /**.:/users**/*principalname* to CDS. This directory can store user-specific application location information. The operation also adds an ACL entry to the default ACL which gives the user **rwtci** permissions on the directory. These permissions allow users to insert objects and links, but they cannot delete the directory or administer replication on the directory. Furthermore, users cannot create additional directories unless you give them **w** (write) access to the clearinghouse. You can read more about the purpose and use of CDS directories in “Managing CDS Directories” on page 1-30. You can read more about ACLs and CDS directories in “Controlling Access to CDS Names.”

Permissions Required — Creating a New User

You generally need numerous permissions to create new users in your DCE cell, so you should log into the cell administrator's account (or a similar privileged account). The **user** reference page lists the required permissions.

Examples — Creating a New User

- To create a new user in a DCE cell, invoke a **user create** operation. The following example creates a principal name **P_Pestana** and an account with the same name. The **create** operation requires your password to prevent someone else from using an unattended session to create an unauthorized account. You must also provide the **–password** option to specify a password for the user. The required **–group** and **–organization** options add principal **P_Pestana** to the named group and organization. The optional –**fullname** option creates a full name to help other human users recognize the principal.

  ```
  dcecp> user create P_Pestana -fullname {Patricia Pestana} -mypwd mxyzptlk \
  > -password change.me -group users -organization managers
  dcecp>
  ```

- You can create multiple users by specifying a list of user names as an argument to the **user create** operation. This method poses some limitations, however. All created users will have the same initial password, group name, and organization name. Furthermore, you cannot specify the **fullname** and

uid attributes because they are unique for each user. The following example creates several users with a password **change.me**, a group name of **users**, and an organization named **staff**:

```
dcecp> user create {R_Lee B_Joy N_Lynn D_Dee} -mypwd mxyzptlk
                  -password change.me -group users -organization staff
dcecp>
```

- Optionally, you can set the **users** or **staff** variables, or both as Double Byte Character Set (DBCS). If you want to use non-PCS characters, see "Introduction — DCE Control Program Usage" on page 2-15.

Creating, Maintaining, and Deleting Keytab Files

Introduction — Creating, Maintaining, and Deleting Keytab Files

The following **dcecp** commands allow you to create, maintain, and delete keytab files:

keytab create Creates keytab files and all their key entries.

keytab delete Deletes keytab files and all their key entries.

keytab add Adds key entries to keytab files.

keytab remove Removes key entries from keytab files.

More Detailed Information — Creating, Maintaining, and Deleting Keytab Files

The Keytab File

Keytabs files are stored on the same machine as the servers whose keys they contain. You can access them remotely and locally using **dcecp**. For remote access, **dcecp** uses **dced** interfaces. The **–local** option to the **dcecp keytab** command allows you to access the local keytab files without using **dced**.

Because **dced** provides remote access to the keytab files, the files are defined as **dced** objects, and those objects are stored in the **dced** controlled portion of the namespace under the **keytab** directory. The **dced** keytab object consists of a UUID to identify the object, an optional annotation, and the name of the file that actually stores the server keys on the local machine. This object is usually a file.

Note that actual server keys are not stored in the keytab object, but in the file stored on the local machine.

The path name of the **dced** keytab object is:

/.:/hosts/*hostname***/config/keytab/***keytab_name*

where:

hostname Is the name of the host on which the **dced** resides.

keytabname Is the name of the keytab file.

The path name to the local keytab file is:

/opt/dcelocal/*keytab_path_name*

keytab_path_name is the path name to the keytab file on the local node.

Protecting Keytab Files

The local keytab files must be adequately protected, and they must not be available on the network. As they are used in the default DCE implementation, the keytab files contain principal keys, which are the basis of DCE security. If these keys are compromised, network security can also be compromised. The calls that access the keytab file use the **rpc_c_protect_level_pkt_privacy**. This protection level performs a DES encryption on the data being passed. The **dcecp keytab –noprivacy** option allows you to specify that your site's default protection level should be used instead.

Create a separate individual keytab file for each server principal that runs on each local node. Servers that share the same keytab file can access each other's keys and thus impersonate each other. Protect the keytab files so that they are readable only by root. If you do this, the servers must be started by root in order to read their keytab files and obtain their key during login.

When you create or change server keys, you can name a different keytab file for each server that runs on the local node. Protect the file so that it is readable only by the server whose key it contains. Then set the **setuid** bit for the server file to the server's identity so that the server can access the keytab file and obtain its key.

Server and Machine Key Version Numbers

When keys are added to the keytab file, each is assigned a version number that ranges from 1 to 255. Whenever server or machine keys change (automatically or explicitly), the key's version number is incremented. Version numbers allow two or more keys to exist for any given server or machine. When keys are changed, any servers or machines that are still using tickets granted under the older unchanged version of the key run without interruption until the ticket expires naturally. When the ticket expires, the server or machine reauthenticates and obtains the new key.

If you use the **–registry** option to the **keytab add** command, old keys are automatically deleted, if possible. If you do not use this option, you should occasionally list the contents of the keytab file by using the **keytab list** command, and use the **keytab delete** command to delete any old versions that are obsolete.

Note: Take care when you are deleting keys from the keytab file. When principal keys are changed, tickets can exist that are based on the key that you deleted. If you delete a key from the keytab file, any active tickets that are based on the deleted key will not be accepted by servers, and clients holding those tickets will get authentication failures.

Creating and Maintaining Keys and Keytab Files

Two commands allow you to create key entries:

keytab create Creates keytab files, the keytab file entries, and the **dced** keytab object.

keytab add Adds key entries to existing keytab files.

When you run both commands you supply the name of the keytab file to either create or modify.

The following lists the other options you can supply to the **keytab create** and **add** commands.

Option	Meaning
–local	This option allows you to access the keytab file without using **dced**.
–noprivacy	This option allows you to specify that the protection level to be used should be the default protection level for your site instead of **rpc_c_protect_level_pkt_privacy**.
–member *name*	The name of the principal (server or machine) whose key you are creating or changing. You can supply multiple names in a list. If you supply a list, all principals named in the list are assigned the same key.

–key *key* The plain text key to the account. This option cannot be used with the **–random** option.

–random This option generates a random key. If you use this option, you must also use the **–registry** option to add the randomly-generated key to the server's or machine's account in the registry. This option cannot be used with the **–key** option.

–registry This option updates the principal's key in the registry to match the key that you enter (or generate automatically) for the key in the keytab file. Use it to ensure that the principal's key in the registry and the keytab file are synchronized when you change a principal's key in the keytab file.

You must use this option if you use the **–random** option. To use this option, you may need to run the **dcecp login** command to ensure that your network identity is appropriate for modifying the registry database.

–version *number* This option specifies a version number for the key. It is required if you do not use the **–registry** option.

–storage *local_file_name*
The path name of the local file to be created. This option is used only for the **keytab create** command. When you add entries to an existing keytab file, you identify the file by its **dced** object name.

–data *keys* The server principal name and keys in the format *principal_name* **key_type {version} {**key_value**}**

Creating a Keytab File

Use the **keytab create** command to create keytab files, entries in the files, and the corresponding **dced** object. When you use this command, you must supply the path name of the **dced** object to be created as an argument, the **storage** option to specify the keytab's local, the **data** option to specify the name of the server principal and the keys, and any of the appropriate options listed above.

This **data** option is in the form:

principal_name key_type **{***version***} {***key_value***}**

where:

principal_name Is the name of the server principal for which the keytab file is being created.

key_type Is a code that specifies whether the key is stored in plain text or in DES encrypted format:

- **des** indicates DES encryption
- **plain** indicates plain text

version Is the key's version number. If you supply no version number the key is assigned a number of 1.

key_value Unless you specified the **–random** option to randomly generate keys, you must supply a key value. If *key_type* is **plain**, you supply a the key plain text. If *key_type* is **des**, you must supply a DES encrypted key.

The following sample command performs the tasks listed below:

- Creates the **dced** keytab object named /**.:/hosts/music/config/keytab/svr4_key**
- Creates the keytab file named /**opt/dcelocal/**keys/svr4_key** in the **keys** directory on the local machine named **music**
- Creates an plain text key entry in the file for principal **mahler** and assigns it a version number of 3.

```
dcecp> keytab create /.:/hosts/music/config/keytab/svr4_key
                     -attr {{storage /opt/dcelocal/keys/svr4_key}
                     {data {mahler plain 3 mon#Repos}}
```

Adding Entries to a Keytab File

Use the **keytab add** command to add entries to an existing keytab file. When you use this command, you must supply the name of the keytab file's **dced** object and any of options described earlier.

You can remove entries from a keytab file by using the **dcecp keytab remove** command. When you use this command, you must supply the name of the keytab file's **dced** object.

When you use the **keytab remove** command, you must supply the name of the keytab file and the name of the principal (or a list of principals) for which to delete keys.

You can also supply the **–version** option to specify the version number of the key or keys to be deleted and the **–type** option to specify the type of keys to be deleted (**plain** for plain text keys or **des** for DES encrypted keys). If you use the **–version** or **–type** options, only keys of the specified version or type will be deleted.

You can remove local and keytab files and their associated **dced** objects by using the **dcecp keytab delete** command.

To delete the local keytab file and the **dced** object, supply the local file name to the command. You can delete multiple keytab files with one command by enclosing the names in braces and separating them with spaces.

To delete only the **dced** object, use the **–entry** option.

Changing Server and Machine Passwords in the Keytab File

Passwords for all principals must be changed when they expire. Principals can use their platform's **chpass** command to change their password. The **dced** daemon's Security Validation Service automatically changes the machine's password as necessary by assigning a randomly generated password. This daemon is supplied with DCE and runs on each local machine that engages in network access. Generally, you can assume that servers or applications created by other vendors also automatically change their password as required by randomly generating passwords. However, if a server that runs under its own identity does not automatically update its password, you must do it manually by using the DCE control program's **keytab add**.

Note: Servers that run under the identity of a human principal should not automatically update their own passwords. When such a server updates its password, it also updates the password of the human principal under whose identity it runs. The human principal must then supply this randomly generated password to log into the system and to reauthenticate. Since the human principal can never know the randomly generated password, the principal cannot log into the system and cannot reauthenticate.

Steps — Creating, Maintaining, and Deleting Keytab Files

If a server's or machine's password is compromised, you must change it in the registry and in the server's local keytab file by performing the following steps:

1. Use the **keytab remove** command to delete the compromised password.
2. Use the **keytab add** command to create a new password for the server or machine.

3. If you do not use the **registry** option of the **keytab add** command to update the server's or machine's registry account simultaneously with the server's or machine's keytab file, run the **registry modify** command to change the server's or machine's password in the registry to match the one in the keytab file.

Examples — Creating, Maintaining, and Deleting Keytab Files

- The following command adds a key to the keytab file named **kfile_3** for the server principal **svr_3**. The key is generated automatically, and the registry is updated to be synchronized with the keytab file.

```
dcecp> keytab add /.:/hosts/foo/config/keytab/kfile_3 \
            -member svr_3 -random -registry
```

- The following command removes all DES keys for the principal **svr_2** in the keytab file /**.:/hosts/foo/config/keytab/kfile_3**:

```
dcecp> keytab remove /.:/hosts/foo/config/keytab/kfile_3
                     -members svr_2 -type des
dcecp>
```

- The following command deletes the keytab files and the **dced** objects named /**.:/hosts/foo/config/keytab/kfile_2** and /**.:/hosts/foo/config/keytab/kfile_3**.

```
dcecp> keytab delete {/.:/hosts/foo/config/keytab/kfile_2
                     /.:/hosts/foo/config/keytab/kfile_3}
dcecp>
```

- The following command removes the **dced** object named /**.:/hosts/foo/config/keytab/kfile_3**, but leaves the local file /**opt/dcelocal/keys/kfile_3** untouched.

```
dcecp> keytab delete -entry /.:/hosts/foo/config/keytab/kfile_3
dcecp>
```

Deleting a User

Introduction — Deleting a User

When users leave your organization, you might need to delete the user from the cell. Use the **user delete** command to do this. This operation removes the principal name from the registry which, in turn, deletes the account and removes the principal from any groups and organizations. The operation also deletes the /**.:/users**/*principalname* directory and any contents from CDS.

Permissions Required — Deleting a User

You need numerous permissions, such as those generally associated with cell administrator, to delete a user. See the **user** object reference page in the *DCE for AIX Administration Command Reference*.

Examples — Deleting a User

- The following example operation removes user **P_Pestana** from the cell:

```
dcecp> user delete P_Pestana
dcecp>
```

- You can remove multiple users from your cell by specifying a list of user names as an argument to the **user delete** operation, as follows:

```
dcecp> user delete {W_Rosenberry J_Hunter P_Pestana}
dcecp>
```

- If you have permissions in a foreign cell, you can remove one or more users from that cell by specifying the global principal name of the users to be deleted. For example:

```
dcecp> user delete /.../their_cell.goodco.com/J_Jones
dcecp>
```

Displaying All Configured DCE Servers and DCE Hosts

Introduction — Displaying Configured Servers and Hosts

Some DCE cells may be relatively stable, with few DCE hosts or DCE servers being added or removed. Other cells can be quite dynamic, with hosts and DCE servers being added, removed, or moved weekly or even daily. In this environment, tracking the locations of DCE resources can be difficult, so the **cell** task object has a **show** operation that scans various databases in the cell returning the names of configured DCE servers and DCE hosts.

More Detailed Information — Displaying Configured Servers and Hosts

One use of a **cell show** command could be to track performance problems. For example, maybe many new hosts and users have been added, but the number or location of CDS or security servers has not grown accordingly. For example, if you have just been hired to administer a new cell, you may need to identify the contents of your cell.

To show configured DCE servers and hosts in a cell, enter a **cell show** operation. The command returns a list of servers grouped by type, along with a list of DCE hosts, as follows:

secservers Each value is the name of a security server.

cdsservers Each value is the name of a machine running a CDS server. The name is the simple name found under /**.:/hosts**. A clearinghouse must be configured on that machine.

dtsservers Each value is the name of a DTS server in the cell.

hosts Each value is the name of a host in the cell, including machines mentioned previously as servers. This is simply the return value of a **directory list /.:/hosts** operation.

Examples — Displaying Configured Servers and Hosts

- The following example shows the names of all the configured DCE servers and hosts in the local cell:

```
dcecp> cell show
{secservers
 /.../my_cell.goodco.com/subsys/dce/sec/master}
{cdsservers
 /.../my_cell.goodco.com/hosts/earth}
{dtsservers
 /.../my_cell.goodco.com/hosts/krypton}
{hosts
 /.../my_cell.goodco.com/hosts/earth
 /.../my_cell.goodco.com/hosts/jupiter
 /.../my_cell.goodco.com/hosts/krypton
 /.../my_cell.goodco.com/hosts/mars
 /.../my_cell.goodco.com/hosts/mercury
 /.../my_cell.goodco.com/hosts/neptune
 /.../my_cell.goodco.com/hosts/pluto
 /.../my_cell.goodco.com/hosts/saturn
 /.../my_cell.goodco.com/hosts/uranus
 /.../my_cell.goodco.com/hosts/venus}
dcecp>
```

- If you have the necessary permission, you can show the configured DCE servers and hosts in another cell by including that cell's name as an argument as shown in the following example:

```
dcecp> cell show /.../their_cell.goodco.com
{secservers
 /.../their_cell.goodco.com/subsys/dce/sec/master}
{cdsserver
 /.../their_cell.goodco.com/gold}
{dtsservers
 /.../their_cell.goodco.com/hosts/silver/dts-entity}
{hosts
 /.../their_cell.goodco.com/hosts/brass
 /.../their_cell.goodco.com/hosts/bronze
 /.../their_cell.goodco.com/hosts/copper
 /.../their_cell.goodco.com/hosts/gold
 /.../their_cell.goodco.com/hosts/iron
 /.../their_cell.goodco.com/hosts/mercury
 /.../their_cell.goodco.com/hosts/silver
 /.../their_cell.goodco.com/hosts/steel
 /.../their_cell.goodco.com/hosts/tin}
dcecp>
```

Displaying All Servers Configured for a DCE Host

Introduction — Displaying All Servers

In larger cells, in which DCE servers and application servers reside on multiple hosts, you will likely want to see what servers are configured to run on particular hosts from time to time. The DCE control program's **host show** operation reads a DCE host's server configuration and execution information returning a list of configured servers on that host. The list contains each server's simple name and indicates whether it is running. The list also indicates whether a security server is a master or replica and whether a DTS entity is a clerk or server.

This operation relies on the **server** object (and consequently on the DCE host daemon) to show information about configured servers. You can read more about controlling server operation in “DCE Application Administration” on page 1-27.

Example — Displaying All Servers

The following example shows the servers configured to run on DCE host **xoltar**:

```
dcecp> host show /.:/hosts/xoltar
video_clip running
dts-entity running clerk
dcecp>
```

Displaying User Information

Introduction — Showing User Information

Sometimes you might want to view the attributes for a user. For instance, you might want to see the expiration date for one or more accounts or view the full name of a principal.

The **user show** command returns the attributes associated with users that are included as arguments to the command. The attributes include principal attributes and ERAs, and account attributes and policies. The information is returned as if the following commands were run in the following order:

- **principal show**
- **account show –all**

Example — Showing User Information

The following command shows the principal and account attributes associated with user **P_Pestana**:

```
dcecp> user show P_Pestana
{fullname {Pat Pestana}}
{uid 5139}
{uuid 00001413-ad4f-21cd-8c00-0000c08adf56}
{alias no}
{quota unlimited}
{groups users}
{acctvalid yes}
{client yes}
{created /.../my_cell.goodco.com/cell_admin 1994-08-01-16:41:32.000+00:00I-----}
{description {}}
{dupkey no}
{expdate none}
{forwardabletkt yes}
{goodsince 1994-08-01-16:41:32.000+00:00I-----}
{group users}
{home /}
{lastchange /.../my_cell.goodco.com/cell_admin 1994-08-01-16:41:32.000+00:00I-----}
{organization managers}
{postdatedtkt no}
{proxiabletkt no}
{pwdvalid yes}
{renewabletkt yes}
{server yes}
{shell {}}
{stdtgtauth yes}
nopolicy
dcecp>
```

You can show information about multiple users by specifying a list of user names as an argument to the **user create** operation.

Listing the DCE Hosts in a Cell

Introduction — Listing DCE Hosts

You can determine the number and names of DCE hosts configured in your DCE cell by using the **host catalog** operation. This operation might be useful for determining whether a specific host has already been configured into your cell. The host does not have to be running for this operation to work because the **host catalog** operation actually performs a **directory list** /**.:/hosts** operation and does not interact with the host. This method relies on the convention that hosts register their names in the /**.:/hosts** directory. If your hosts register in some other directory, you need to modify the **host catalog** operation in the **host** task object. You can read more about the purpose and use of CDS directories in “Managing CDS Directories” on page 1-30.

More Detailed Information — Listing DCE Hosts

The **host catalog** operation resembles the **cell show** operation except it does not separately list DCE servers.

Examples — Listing DCE Hosts

- The following example operation lists all DCE hosts that have been configured in the cell:

```
dcecp> host catalog
/.../my_cell.goodco.com/hosts/bigbox
/.../my_cell.goodco.com/hosts/drifter
/.../my_cell.goodco.com/hosts/duh
/.../my_cell.goodco.com/hosts/heater
/.../my_cell.goodco.com/hosts/pc1
/.../my_cell.goodco.com/hosts/pc2
/.../my_cell.goodco.com/hosts/pc3
/.../my_cell.goodco.com/hosts/peewee
/.../my_cell.goodco.com/hosts/xoltar
/.../my_cell.goodco.com/hosts/xray
/.../my_cell.goodco.com/hosts/zoof
dcecp>
```

- You can omit the cell name by using the **–simplename** option as in the following example.

```
dcecp> host catalog -simplename
hosts/bigbox
hosts/drifter
hosts/duh
hosts/heater
hosts/pc1
hosts/pc2
hosts/pc3
hosts/peewee
hosts/xoltar
hosts/xray
hosts/zoof
dcecp>
```

Modifying or Extending the Cell Object

The **cell** task object is implemented as a script so that administrators can modify or extend it on a per-site basis. Here are a few examples of possible modifications or extensions you can make.

- Add a way to show GDS or DFS server information.
- Add options to the **cell show** operation to omit listing all the hosts in a cell or to show only certain DCE servers.

Modifying or Extending the Host Object

The **host** task object is implemented as a script so that administrators can modify or extend it on a per-site basis. For example, administrators might want to add GDS and DFS information to the object. You could also add calls to specialized commands to start or stop application servers. For instance a **printer stop** operation could be useful.

Modifying or Extending the User Object

The **user** task object is implemented as a script so that administrators can modify or extend it on a per-site basis. For example, administrators might want to add GDS and DFS information to the object. Other possible modifications include the following:

- Changing the location of the CDS directory created for users, or removing it completely.

- Changing the default ACLs placed on the various objects.
- Add an option to give users write access to the clearinghouse where the master replica of the /**.:/users**/*username* directory resides. This allows users to create their own subdirectories. The option could add individual principal names to the clearinghouse ACL. An easier method could add principals to a group that has write access to the clearinghouse.
- Setting certain attributes or policies on all newly created principals and accounts to match the site's policies. For example, you could set principals to have a **pwd_val_type** ERA and set accounts to generate random passwords.
- Setting up site-specific defaults for passwords (to be changed by the user later), groups, organizations, principal directories, and so on.
- Supporting a **user modify** command. Such a command could change group or organization information or some other attributes associated with users.

Removing a DCE Host from a Cell

Introduction — Removing a DCE Host from a Cell

Occasionally, you might want to remove a DCE host from a cell. For instance, your organization is replacing some older systems that are being sold to another organization.

Removing or unconfiguring a DCE host is more than just erasing DCE information from the host's disk because CDS and the DCE Security Service both maintain host-specific information that needs to be removed as well. The **host unconfigure** operation deletes all objects, directories, and links from the /**.:/hosts**/*hostname* CDS directory including the directory itself. It also deletes all principal names beginning with **hosts**/*hostname*/ which, in turn, removes all accounts with the same name. Finally, it removes all local configuration files and stops all running DCE processes ending with the DCE daemon (**dced**).

More Detailed Information — Removing a DCE Host from a Cell

The **host unconfigure** operation operates only on remote hosts. You cannot perform this operation on a local host because it removes the DCE Security Service registry information needed to complete the operation. Also note that you need cell administrator privileges to perform a **host unconfigure** operation.

To remove a remote DCE host from a cell, use a **host unconfigure** operation providing the DCE host name of the host to be unconfigured.

You can remove a multiple DCE hosts from a cell by providing a **dcecp** list of names of the hosts to be unconfigured.

If you have cell administrator privileges in a foreign cell, you can remove a remote DCE host from that cell by supplying a global DCE name of the host to be unconfigured.

Examples — Removing a DCE Host from a Cell

- The following example removes host **calypso** from the cell:

```
dcecp> host unconfigure /.:/hosts/calypso
dcecp>
```

- The following example removes hosts **calypso**, **hydra**, and **nova** from the cell:

```
dcecp> host unconfigure {/.:/hosts/calypso /.:/hosts/hydra /.:/hosts/nova}
dcecp>
```

- The following example removes host **gobo** from foreign cell /**.../their_cell.goodco.com**:

```
dcecp> host unconfigure /.../their_cell.goodco.com/hosts/gobo
dcecp>
```

Setting Up a Password Synchronization Server

Introduction — Setting Up a Password Synchronization Server

The Password Synchronization Server (PSS) can be installed and configured by the IBM Install/Configure GUI on any IBM DCE for AIX machine in a cell. When one or more user accounts are configured to use this server, password changes for these accounts are automatically propagated to one or more foreign registries that have been configured to receive these changes. A *foreign registry* is defined as any DCE server that exports the password propagation interface (that is, the **rsec_pwd_propagate** API). See the *DCE for AIX Application Development Guide — Core Components* and the *DCE for AIX Application Development Reference* for more information.

The PSS also supports foreign registry on-demand requests for the password associated with a specific account. A request is made by using the **sec_rgy_attr_lookup_by_name** or **sec_rgy_attr_lookup_by_id** APIs to read the Extended Registry Attribute (ERA) named **PLAINTEXT_PASSWORD** that has been associated with that account. (This association takes place when the account is configured for password synchronization support.)

The PSS is typically useful for customer environments in which one or more legacy systems have not yet been fully integrated with DCE. Such systems often maintain their own registry of user definitions, including the passwords associated with each definition. When these user definitions also appear in the DCE Security registry, it is sometimes desirable to keep the passwords synchronized between the DCE Registry and the foreign registry. Because the password in a foreign registry is often encrypted with a different algorithm than the one used by the DCE Registry, and because the latter is irreversibly encrypted, a foreign registry must have access to the plain text version of the password in order to use its own encryption algorithm. The PSS provides access to this plain text version.

More Detailed Information — Setting Up a Password Synchronization Server

Configuration of the PSS results in the creation of the following registry schema definitions:

- **PW_PROPAGATE_RETRY_INTERVAL**
- **PW_PROPAGATE_ENABLE**
- **AVAILABLE_STRENGTH_SERVER**
- **FOREIGN_REGISTRY**
- **PASSWORD_STRENGTH**
- **SECONDARY_PWD_MGMT_BINDING**
- **FOREIGN_REGISTRY_PW_PROP_BINDING**
- **PLAINTEXT_PASSWORD**

Additionally, instances of these and other schemas are attached to certain principals as follows:

- When PSS is configured, **PW_PROPAGATE_RETRY_INTERVAL**, **PW_PROPAGATE_ENABLE** and **PWD_MGMT_BINDING** are associated with the PSS principal.
- When a foreign registry is configured:

  - An instance of **FOREIGN_REGISTRY** is associated with the PSS principal.
  - An instance of **FOREIGN_REGISTRY_PW_PREP_BINDING** is associated with the foreign registry server's principal.
- When an account is configured for PSS support, instances of **PWD_MGMT_BINDING**, **PWD_VAL_TYPE**, **FOREIGN_REGISTRY**, and **PLAINTEXT_PASSWORD** are associated with the account principal.
- When an account is configured for both PSS and password strength support:
  - An instance of **PASSWORD_STRENGTH** is associated with the account's principal.
  - An instance of **AVAILABLE_STRENGTH_SERVER** is associated with the PSS principal.
  - An instance of **SECONDARY_PWD_MGMT_BINDING** is associated with the password strength server principal.

Installation and Usage Considerations

- The PSS must be installed on a machine running IBM DCE for AIX.
- The PSS must be configured before any foreign registry is configured and before any account is configured for PSS support.
- All foreign registries configured for support by the PSS must reside in the same cell as the PSS.
- Though not a requirement, it is recommended that the PSS reside on the same machine as the DCE Security server.

Operation

For each account configured for password synchronization, the PSS will propagate account password changes to one or more foreign registries configured for that account. If a foreign registry does not respond (for example, because of netware outage or a foreign registry out-of-service condition) or returns an error, the PSS will retry propagation at regular (configurable) intervals until successful. Similarly, if the PSS goes out of service without successfully propagating outstanding requests, it will “remember” these requests when restarted and will resume propagation attempts.

Steps — Setting Up a Password Synchronization Server

To make use of the PSS, three operations must be completed:

1. Install and configure the PSS. You can accomplish this through the IBM Install/Configure GUI provided with IBM DCE for AIX.
2. For each foreign registry requiring password synchronization, configure that registry for this function. This must be accomplished by a customer-provided procedure that includes the following steps:
   a. Use **rgy_edit** or **dcecp** to create an account for the foreign registry. (In the remaining steps of this procedure, this account is referred to as *princname*.)
   b. Use **cdscp** or **dcecp** to create the following directory in the CDS namespace:

      ```
      /.:/subsys/dce/fgnreg
      ```

      Note: You only need to perform this and the following step once regardless of the number of foreign registries being configured.
   c. Use **acl_edit** (without **–e**) or **dcecp** to grant **rwidtca** permissions to your cell administrator (typically, **cell_admin**) for the above-referenced directory.
   d. Use **cdscp** or **dcecp** to create the following entry in the CDS namespace:

      ```
      /.:/subsys/dce/fgnreg/princname
      ```

Note: The foreign registry server must export interface and binding information to this entry during its initialization.

e. Use **acl_edit** (with **–e**) or **dcecp** to grant the following permissions on this entry:

- Give **rwdtc** permission to **cell_admin** (cell administrator).
- Give **rwdt** permission to *princname*.

f. Use **rgy_edit** or **dcecp** to create a key table for *princname* (standard practice for server accounts).

g. Use **dcecp** to attach an instance of the **FOREIGN_REGISTRY_PW_PROP_BINDING** ERA to *princname* with the following values:

```
{{dce princname protect_level secret name} /.:/subsys/dce/fgnreg/princname}
```

Note: The value to be specified for *protect_level* is determined by the location of the machines housing the PSS and foreign registry server, as follows:

- If *both* machines reside in the United States or *both* machines support data privacy (DES), specify **pktprivacy**; otherwise,
- If the foreign registry is housed on a machine running IBM DCE for AIX, specify **cdmf**; otherwise,
- Specify **pktinteg**.

h. Use **dcecp** to attach an instance of the **FOREIGN_REGISTRY** ERA (containing the foreign registry principal name) to the **pwsync** principal, as follows:

```
dcecp> principal modify pwsync -add {foreign_registry princname}
```

Note: When an account is configured for password synchronization (Step 3 below), an instance of the **PLAINTEXT_PASSWORD** ERA is associated with it and a default protection level is included. If all foreign registries specified for the account configuration are housed on machines running IBM DCE for AIX, this default should be left "as is." If one or more foreign registries for any account configured for password synchronization are housed on machines not running IBM DCE for AIX, and if one or more of these machines do not support data privacy (DES), then you must modify the schema definition for **PLAINTEXT_PASSWORD** to specify **pktinteg**.

3. For each account that uses password synchronization, configure the account for this function. You can accomplish this task using the IBM Administration GUI provided with IBM DCE for AIX.

Restrictions — Setting Up a Password Synchronization Server

- If an account is configured for PSS support but not for password strength checking, registry policy checks on password composition (that is, all blanks, at least one non-alphanumeric, minimum length) will not be enforced.
- The PSS cannot be replicated.
- If the DCE Security server and/or a foreign registry reside on machines *not* running IBM DCE, and one or more of these machines do not support data privacy (DES), passwords will not be protected (encrypted) on-the-wire when received by a foreign registry. Otherwise, they will be protected by either DES or IBM's Commercial Data Masking Facility (CDMF).
- If the PSS is out of service, an attempt to change the password of an account that has been configured for password synchronization will fail.

Setting Up a Password Strength Server

Introduction — Setting Up a Password Strength Server

The DCE password format policy enables you to control the following characteristics of user passwords:

- Minimum password length
- Whether a password can be all spaces
- Whether a password can consist of alphanumerics only

These characteristics are validated in the default implementation of a password strength server (**pwdstren.exe**) provided with the IBM DCE for AIX product.

More Detailed Information — Setting Up a Password Strength Server

You can extend the password strength policies in your cell by creating a password management server to perform customized password checking and generation. DCE provides an example password validation/generation server which you can use as the basis for a password management server that suits your cell's requirements. If you have installed EXAMPLES, you can find this code in the **$DCELOCAL/examples/pwdstren** directory. DCE also provides a Password Management API that application developers can use to acquire information about the principal's password management policy, and to request generated passwords from the password management server. See the *DCE for AIX Application Development Guide — Core Components* for information on the Password Management API.

Having created this server, you can then constrain a principal's password to be validated by this server when it is created and whenever it is changed. You do this by using the install/configuration GUI provided with the IBM DCE for AIX product to configure the service and by using the Administrative GUI (Account Administration) provided with the IBM DCE for AIX product to configure one or more accounts to use this server.

Use of the Administrative GUI to configure an account for password strength checking results in attaching of ERAs to the account principal, as follows:

pwd_val_type Specify password creation options for the principals as follows:

0 (**NONE**) Specifies that the principal's password is subject only to DCE standard policy. Specifying **0** (NONE) is equivalent to not attaching an ERA instance to the principal.

1 (**USER_SELECT**) Specifies that the principal must supply password text as input to the password management server specified in the *pwd_mgmt_binding* ERA.

2 (**USER_CAN_SELECT**) Specifies that the principal can either supply password text or use the password management server specified in the *pwd_mgmt_binding* ERA to generate a password for the user.

3 (**GENERATION_REQUIRED**) Specifies that the password management server specified in the *pwd_mgmt_binding* ERA should generate a password for the principal.

pwd_mgmt_binding
Specify a binding to your cell's password management server.

For information on requesting generated passwords when changing a password, see “Generating Passwords Using **dcecp**” in this section. For information on configuring a password management server, see “Managing a Password Management Server” in this section.

Note: If an account is configured for both password strength checking and for password synchronization (see “Setting Up a Password Synchronization Server” on page 3-15), the definition and use of the **pwd_val_type** and **pwd_mgmt_binding** ERAs will differ from the above definitions. For that

reason, you should use the IBM DCE for AIX Administrative GUI (Account Administration) to configure accounts for password strength and/or password synchronization.

Managing a Password Management Server

The following provides some notes on password management server management.

- To protect password security and to optimize performance, the password management server should run on the same machine as the master DCE Security Server.
- The default path name for the password management server is **$DCELOCAL/bin/pwd_strengthd**. You can change this path name using the **PWD_MGMT&SVR** environment variable in **config.env**. If you construct one or more additional servers, you can locate them in the path of your choosing, as the install/configuration GUI prompts you for their location.
- The configuration process supports configuration of multiple password management servers in a cell.
- To replace a password management server that has been configured for use by one or more accounts with another version, see "Steps — Setting Up a Password Strength Server."
- The log file for the default password management server resides in **$DCELOCAL/var/security/pwd_strengthd.log**. This location is built into the server code and is not configurable.

Generating Passwords Using dcecp

If a *pwd_val_type* ERA having the values **2 (USER_CAN_SELECT)** or **3 (GENERATION_REQUIRED)** exists for a principal, that principal can (or will be required to) request a generated password when he changes passwords. If you are the principal **smitty**, the following sequence of **dcecp** commands can be used:

```
dcecp> set p [account generate smitty]
newgenpwd
```

This command requests a generated password from the Password Management server, places the new password in the *p* variable, and prints it to the screen (*newgenpwd*). (Be sure to remember the new password.) Next, pass the value stored in *p* as the value of new password in an **account modify** or **account create** command:

```
dcecp> account modify smitty -password $p -mycurrentpwd -dce-
```

Steps — Setting Up a Password Strength Server

To replace a password management server that has been configured for use by one or more accounts with another version, follow this procedure:

1. Kill the server you want to replace.
2. Rename **$DCELOCAL/bin/***server name*.
3. Copy the new server into **$DCELOCAL/bin/***server name*.
4. Start *server_name*.

Note: This procedure works only if the new server is represented by the same server principal name as the server being replaced. If this is not the case, you must unconfigure password strength for each affected account, install and configure the new strength server, and reconfigure each account for password strength.

Restrictions — Setting Up a Password Strength Server

NEVER execute the following **dcecp** command because the password will be changed in the account but the user will not see the new password:

```
dcecp> acct mod smitty -password [acct gen smitty] -mycurrentpwd -dce-
```

Starting Configured DCE Processes on a Host

Introduction — Starting Configured DCE Processes

Each host's DCE daemon (**dced**) can maintain configuration information for servers set to run on that particular host. This information is established using an application's installation script or by using the **server** object directly. While the **server** object provides its own **start** operation that can start individual servers on a host, you must explicitly name each server. The **host start** operation, which is not supported, lets you start all configured DCE servers and clients and all configured application servers on a host with a single command.

More Detailed Information — Starting Configured DCE Processes

To operate on a remote host, its DCE daemon must be running. Remote **host start** operations also require at least one CDS server and one security server to be running in the cell. The **host start** operation operates on DCE servers and clients and on application servers that are configured by using the **server** object. Application servers must be configured with the **starton** attribute set to **boot** or **explicit**. You can read more about configuring application servers in “DCE Application Administration” on page 1-27.

Example — Starting Configured DCE Processes

The following example starts all configured servers on host **bigbox**:

```
dcecp> host start /.:/hosts/bigbox
dcecp>
```

Note: Host commands are not supported.

Stopping DCE Processes Running on a Host

Introduction — Stopping DCE Processes

Like the **host start** operation discussed in the previous section, the **host stop** operation is more encompassing than a **server stop** operation, however, it is not supported at this time. It lets you stop all DCE processes on a host with a single command rather than issue a separate **server stop** operation for each server. This operation stops application servers, then DCE processes, and finally when stopping DCE processes on the local machine, stops **dced**. You can read more about controlling servers in “DCE Application Administration” on page 1-27.

More Detailed Information — Stopping DCE Processes

To operate on a remote host, its DCE daemon must be running. Remote **host stop** operations also require at least one CDS server and one security server to be running in the cell. The **host stop** operation operates on DCE servers and clients and on application servers that have been configured by using the **server** object.

Example — Stopping DCE Processes

The following example stops all DCE processes and application servers on host **bigbox**:

```
dcecp> host stop /.:/hosts/bigbox
dcecp>
```

Note: Host commands are not supported.

Testing Cell Operation

Introduction — Testing Cell Operation

When client/server communication problems occur, it is easy to suspect that one or more DCE services is not operating in the cell. You can easily test whether a cell's DCE services are running by invoking a **cell ping** operation.

More Detailed Information — Testing Cell Operation

If called with no option, the **cell ping** operation performs a **server ping** operation on the master security server, on the CDS server that has a master clearinghouse, and all the DTS servers in the cell. Use the **–replicas** option to test CDS and Security Service replicas as well as the masters. The **–clients** option tests every DCE host in the cell by looping though the /**.:/hosts** directory in CDS and performing a **host ping**, which is not supported, with each host name as an argument.

Note: Host commands are not supported.

In case of failure, the operation generates an error and returns a list of servers or hosts that could not be contacted. For any successes, the operation returns the message `DCE Services Available`. For successes with the **–clients** option, the message is `DCE Clients Available`.

Examples — Testing Cell Operation

- The following example pings the names of all the configured master DCE servers in the local cell:

  ```
  dcecp> cell ping
  DCE services available
  dcecp>
  ```

- The following example pings the names of all the configured DCE hosts in the local cell. Depending on the size of a cell and timeout values set, this command can take a long time (from several to many minutes) to complete.

  ```
  dcecp> cell ping -clients
  DCE clients available
  dcecp>
  ```

- If you have the necessary permission, you can ping the configured DCE servers and hosts in another cell by including that cell's name as an argument as shown in the following example:

```
dcecp> cell ping /.../their_cell.goodco.com
DCE services available
dcecp>
```

Testing Whether a DCE Host is Running

Introduction — Testing Whether a DCE Host is Running

Because DCE communications often involve several steps before clients communicate with their servers, communication failures can be difficult to diagnose. For instance, a server may not be running on a host or the DCE services may not be currently running, even though the host has been configured into the cell. You can use a **server ping** operation to test whether a server process is running but, if this fails, you might need a way to see if the DCE host is even accessible through the network. The DCE control program's **host ping** operation tests whether a host's DCE services are accessible on the network, returning a **1** if it is and a **0** if it is not accessible.

The **host ping** operation, which is not supported, tests for the presence of the remote host's DCE daemon (**dced**). You can read more about the purpose and use of the **dced** in “Managing DCE Host Services and Host Data” on page 1-28.

Example — Testing Whether a DCE Host is Running

The following example tests whether the **dced** on host **duh** is accessible on the network.

```
dcecp> host ping /.:/hosts/duh
1
dcecp>
```

Note: Host commands are not supported.

Using Task Objects to Simplify DCE Administration

Introduction — Simplifying DCE Administration

Individual DCE control program objects operate on very specific pieces of information in DCE. For example, the **group** object operates solely on security groups in the DCE Security Service registry database. The **group** object enables administrators to create and delete security groups, add and remove members from security groups, rename the groups, and so on. Such precise control is necessary because it allows you to custom tailor DCE to meet very specific needs or circumstances.

While such control might be necessary when configuring a new cell or fixing some access control problem, it can overwhelm routine DCE administration tasks.

More Detailed Information — Simplifying DCE Administration

One of the task objects is the **user** object that you can use to add and remove user information in your DCE environment. For instance, a single invocation of the **user** object can perform all six of the previously mentioned steps needed to correctly add a new user to your DCE environment. You can also use this same task object to delete the user from your environment.

The task objects are implemented as **dcecp** scripts by using the DCE control program language, which means that you can extend the scripts or change their behavior according to your needs. For instance, the default implementation of the **user** task object does not operate on any GDS or DFS information. If

your DCE environment includes these extended services, you might want to add some GDS or DFS operations to the script.

Steps — Simplifying DCE Administration

As an example, look at the minimum steps needed to add a new user to a DCE cell:

1. Use the **principal** object to create a principal name for the user.
2. Use the **group** object to add the principal to a security group.
3. Use the **organization** object to add the principal to a security organization.
4. Use the **account** object to create an account for the principal.
5. Use the **directory** object to create a directory for the principal in CDS.
6. Use the **acl** object to give the principal access to the CDS directory.

Performing these six steps probably would not pose any problems in a small cell with 15 or 20 users. But consider a cell with more, perhaps a hundred or maybe even a thousand or more users, and the need to automate this and other administration tasks becomes evident.

To meet this administration need, the DCE control program includes several administration task objects for performing some routine DCE administration tasks. Here, we are using the term *task* to mean doing something that requires multiple steps, such as when adding a user consists of performing six lower-level operations.

Chapter 4. DCE Host and Application Administration

This section describes host and application administration within DCE. For introductory and overview information on host and application administration, see “DCE Application and Host Administration Overview” on page 1-27. That section also groups related tasks together in the order in which they may normally occur so you can access these referenced tasks in a logical, tutorial-type manner. You may find this helpful in performing related tasks together from beginning to end.

Client Administration

Introduction — Client Administration

This section discusses the administration needs of application clients. Although client administration is very simple — there are just two related operations — it is an essential step in getting clients and servers working together.

We know that CDS is a hierarchical system of directories that stores server binding information in the form of server entries. We also know that CDS offers group entries and profile entries as a way to direct clients to appropriate servers. But how do clients know where to begin looking for a server?

More Detailed Information — Client Administration

Servers register interfaces and their bindings in CDS. Each interface-binding combination is registered under a server entry name. When a client makes a remote procedure call, it passes a server entry name (or a group or profile entry name) to CDS along with the UUID of an interface that offers the remote procedure. CDS uses the server entry name (or group or profile entry name) as a starting point in the search for a binding that contains an interface UUID and version matching that passed by the client. This method presumes the client has previously acquired the server entry name (or group or profile name) used by the server.

Getting clients to use an appropriate server entry name is a two step process:

1. Determine what entry name a client should use.
2. Pass the name to the client program.

Note that a client uses whatever name you supply. The client program cannot distinguish whether the name is a server entry name or group entry name or profile entry name. To the client, all of these names look and behave the same.

Determining the Entry Name

You need to know the entry name exported by a server so you can provide it to client programs when you configure them. Here, we are just calling this name an entry name, but it can be a server entry name or group entry name or profile entry name. Your application documentation should help you decide which kind of entry to use.

If you are installing and configuring the server and client parts of an application, make a note of the server's entry name when you configure the server.

If you are not installing or configuring the server (for instance, the server was previously installed), you might need to do some detective work to determine the name to use. There are several places you can look.

If a server uses the server control facility described in “Controlling Server Operation” on page 4-3, you can probably use a **dcecp server show** operation to reveal its entry name. Of course, this means you need to know the server's object name on the host where the server resides. You can see all of the server object names on a host using a **server catalog** operation.

When the server side does not easily reveal its entry name, try to determine what entry other client programs are using. Client programs frequently start from a boot program or script of some kind and entry names are generally provided as arguments to the command to start the client. These commands often follow the same model shown in the previous example of the server startup command.

Providing the Entry Name to Clients

Sometimes, very simple clients can have the server entry name encoded within them so you do not have to pass any entry name. But more often, you need to supply an entry name to a client program when it starts. This approach is more flexible than hardcoding an entry name because it offers an easy way to use a different entry name should the need arise.

The client configuration documentation should include instructions on how to pass the name to the client. One method uses a script or batch file that contains the command to start the client along with arguments that include the appropriate server entry name.

Examples — Client Administration

- The following example lists all the server objects configured on host silver. The **server show** operation reveals the entry name used by the **info_server** program.

  ```
  dcecp> server catalog /.:/hosts/silver
  /.../my_cell.goodco.com/hosts/silver/config/srvrconf/video_clip
  /.../my_cell.goodco.com/hosts/silver/config/srvrconf/info_server
  dcecp> server show /.:/hosts/silver/config/info_server
  {uuid 6d5e7184-71b7-11cd-a205-08000925634b}
  {program {/usr/local/bin/infosrv}}
  {arguments {-brief}}
  {prerequisites {}}
  {keytabs {}}
  {entryname {/.:/subsys/applications/info_server_1}}
  {services {}}
  {principals {}}
  {starton {explicit failure}}
  {uid 1423}
  {gid 1000}
  {dir {/tmp}}
  dcecp>
  ```

  If a server starts from a boot program or script of some kind, look in the program or script for the name or names (sometimes servers use multiple names when they export multiple interfaces). The name might be supplied as an argument to the command that starts the server as in the following example.

  ```
  infosrv /.:/finance/operations/infoserv
  ```

- The following example shows a server entry name passed as a command argument in a shell script that starts the client.

  ```
  # Shell Script to start the InfoClient application
  infoclient /.:/finance/operations/InfoServ_profile
  ```

  Alternatively, the server entry name can be stored in an environment variable (called RPC_DEFAULT_ENTRY). The following example shows a shell script that defines this variable and then invokes the client.

```
#! /bin/sh
# Shell Script to start the InfoClient application
export RPC_DEFAULT_ENTRY=/.:/finance/operations/InfoServ_profile
infoclient
```

Controlling Server Operation

Introduction — Controlling Server Operation

The conventional notion of a DCE application server assumes that a server is running, waiting for client requests to service. While this is an effective model for some general server operations, it does not offer the flexibility needed by DCE applications. Commercial environments will likely have many kinds of servers. Some may need to be constantly available while others may be needed only at certain times of the day. Still others may be needed on an infrequent or unpredictable basis.

An application programmer or administrator could solve these kinds of problems by writing a script or application that monitors server operation, automatically starting or restarting servers when necessary. Such solutions frequently rely on host utilities like startup and shutdown programs or schedulers like **cron**. However, this often requires administrators to log into separate system administration accounts on each host. Moreover, this approach places more burden on developers and administrators to devise independent server control mechanisms which may not be portable, especially in heterogeneous environments.

DCE solves some of these problems by providing a server control facility which offers a variety of ways to control DCE application servers. The server control facility is part of the DCE daemon (**dced**) so servers can rely on it wherever the **dced** runs. Additionally, the facility's administration functions are accessible through **dcecp** so administrators can use consistent (portable) methods to manage servers from any host where **dcecp** is available. Furthermore, access to the server control facility is authenticated, preventing unauthorized or accidental tampering of server control information.

More Detailed Information — Controlling Server Operation

Common Server Configuration Needs

Before you configure a server, you might need to perform some preliminary steps. If a server uses DCE authentication and authorization, its principal name must be registered with the DCE Security Service or run under the DCE identity of the parent process. For details on creating server accounts, see “Creating and Maintaining Accounts” on page 1-48.

Naming Server Configuration Information

Server configuration information is accessible using a name of the form:

/.../*cellname*/**hosts/***hostname*/**config/srvrconf/***servername*

If you have the necessary permissions, you can use the global name to access the configuration database on a remote host (even a host in another cell).

Server Configuration Information

Each DCE for AIX Version 2.1 has a database that can store configuration information for servers on that host. Use the DCE control program **server** object to store, modify or remove server configuration information in the server configuration database on the host system.

You need to specify some or all of the following information when managing server configuration:

uuid An identifier for the particular server configuration object.

program The name (including the path name) that invokes the server program.

directory The name of the program's working directory. When a server is running, it may need a place to store its output or temporary files.

arguments Command line arguments used to start the server.

entryname The name of an RPC entry to which the server exports its binding.

keytabs A list of one or more UUIDs of related keytab objects (files) where the server stores its keys. This information is needed for servers that use DCE authentication or authorization.

principals A list of one or more principal names for the server that are registered in the DCE Security Service. This information is needed for servers that use DCE authentication or authorization.

services Identifies the services offered by the server. Each service attribute consists of an attribute list with the following elements:

- **annotation** A human-readable string describing the service.
- **ifname** The interface name of this service (specified in the interface definition file).
- **interface** The interface identifier (UUID and version number) of this service (specified in the interface definition file).
- **binding** A list of string bindings identifying this service.
- **entryname** The name of an RPC entry to which the server exports its binding for this service.
- **flags** A list of keywords to identify flags for this server. Only the **disabled** flag is currently supported.
- **objects** A list of object UUIDs supported by this service.

uid A POSIX uid that the server is started with.

starton Specifies server starting conditions. The value is a list of one or more of the following:

- **auto** The server starts whenever a request for its service is received by the DCE daemon.
- **explicit** The server starts (or stops) whenever an administrator performs a **server start** or **server stop** operation that directly names the server.
- **boot** The server starts whenever the host system starts.
- **failure** The server starts whenever it has exited with an unsuccessful exit status.

Configuring Servers

Use the **server create** operation to make an application server accessible to the server control facility. Configuring a server means creating the information needed to start and control the server. Typically this includes a server's starting command line and arguments, along with other information needed to start DCE applications.

Some servers need to be available whenever a host system is running. For instance, you might want a server that provides information on host activity to start at the host boot time and run until the host shuts down. Other kinds of services might be needed or only for brief periods. The server control facility has an administrative interface that lets you specify some conditions for starting and stopping servers:

Explicit You can set a server so that you can explicitly start it whenever you want.

Boot You can set a server to start at boot time.

Automatic You can set a server to start on demand; that is, it starts whenever a client request for its services is received at the host system.

Failure You can set a server to start automatically if it exits unexpectedly.

Listing and Retrieving Server Configuration Information

When you want to see a list of the names of servers configured on a particular host, use a **server catalog** operation. This operation does not show every server available on a host, just those that have configuration information stored in the server configuration database.

```
dcecp> server catalog /.:/hosts/silver
/.../my_cell.goodco.com/hosts/silver/config/srvrconf/video_clip
dcecp>
```

List the names of all the configured servers in a DCE cell by using a **foreach** command to repeat the **server catalog** operation for each host in a cell.

```
foreach h [directory list /.:/hosts]{
   echo [server catalog $h]
}
```

If you are unsure of the configuration information established for a server, you can view it using a **server show** operation. Use the **–executing** option to view information about a running server.

```
dcecp> server show /.:/hosts/silver/config/srvrconf/video_clip
{uuid d860322b-d499-11cd-9dfb-0000c08adf56 1.0}
{program {/usr/local/bin/vclip}}
{arguments {-catalog}}
{prerequisites {}}
{keytabs {683cf29a-e456-11cd-8f04-0000c08adf56}}
{services {{annotation "Video Clip Catalog and Server"}}
{principals {Vclip_Srv_1}}
{starton {explicit failure}}
{uid 1441}
{gid 1000}
{dir {/tmp}}
dcecp>
```

Unconfiguring Servers

You can remove server configuration information from a host's configuration database using a **server delete** operation. You would perform this operation for instance, when a server moves to a different host. A **server delete** operation does not stop a server that is currently running.

Note: When the machine containing the master CDS replica is not your local machine, you must perform an additional step. If you unconfigure a stand-alone client, security replica, or CDS replica locally, you must also update the configuration information on the machine containing the master CDS replica to indicate that the client, security replica, or CDS replica has been removed. See “Modifying a Directory's Replica Set” on page 5-50 for more information about removing references to a CDS replica that has been unconfigured locally. Also see *DSS for AIX Up and Running!* for additional information on configuring and unconfiguring servers.

Starting and Stopping Servers

After a server has been appropriately configured, you can use a **server start** or **server stop** operation to start or stop the server remotely.

Disabling and Enabling Services

You can prevent clients from using a service offered by a server — even when the server is running — setting its services to disabled. When set to disabled, server endpoint information is not returned to requesting clients, thereby preventing clients from finding servers. Instead, clients receive a server status of `EPT not registered`. Clients that previously acquired the server endpoint can still communicate with the server, however.

When a server provides multiple interfaces, you can disable any one or more of its interfaces by specifying their interface identifiers.

Extending Server Configurations

Some servers may require configuration information that is not supported by the set of attributes provided with your DCE software. You can add arbitrary information to your server configuration information by creating additional *extended registry attributes* ERAs with the **xattrschema** object.

After creating a new attribute, use a **server modify** operation as explained in "Changing Server Considerations" in this section to insert the necessary data.

ERAs in server configuration information are protected by two levels of ACLs. One ACL type protects the container in which the ERA resides. The second ACL type protects the individual ERA.

The ERA container ACL is named:

/.../*cellname*/**hosts/***hostname*/**config/xattrschema**

The ERA container ACL has the following permissions:

c (control) Modify the container ACL

r (read) Read ERA in the container

i (insert) Create new ERA information

l (Insert) Although the **l** permission is present, it does not apply to ERA items. This permission applies to server control facilities.

The ACL for an individual ERA is named:

/.../*cellname*/**hosts/***hostname*/**config/xattrschema/srvrconf/***ERA_name*

ACLs on individual ERAs can prevent unauthorized principals from creating, reading, changing, or deleting ERA information.

Changing Server Configurations

Sometimes you might want to change a server's configuration information. For instance, you want to change the **–starton** attribute from **boot** to **explicit** so that you can control the server manually.

To change the normal server configuration attributes you must first delete all of the existing attributes and then create new ones. Avoid losing the current information by first using a **server show** operation to display it on your screen.

The following uses a **server show** operation to capture the current server configuration information. The **server delete** operation removes the configuration information and a **server create** operation inserts the new **–starton** attribute along with the remaining server configuration information.

```
dcecp> server show /.:/hosts/silver/config/srvrconf/video_clip
{uuid d860322b-d499-11cd-9dfb-0000c08adf56 1.0}
{program {/usr/local/bin/vclip}}
{arguments {-catalog}}
{prerequisites {}}
{keytabs {683cf29a-e456-11cd-8f04-0000c08adf56}}
{services {{annotation "Video Clip Catalog and Server"}}
{principals {Vclip_Srv_1}}
{starton {explicit failure}}
{uid 1441}
{gid 1000}
{dir {/tmp}}
dcecp> server delete /.:/hosts/silver/config/srvrconf/video_clip
dcecp> server create /.:/hosts/silver/config/srvrconf/video_clip
                    -program /usr/local/bin/vclip
                    -directory /tmp -arguments {-catalog}
                    -principal Vclip_Srv_1
                    -entryname /.:/subsys/applications/video_clip_1
                    -starton {explicit failure}
                    -services {{annotation "Video Clip Catalog and Server"}}
dcecp>
```

You can directly change extended registry attribute information using a **server modify** operation. The following example changes a server's extended registry attribute called **objfamily** to contain new values. This operation assumes the extended registry attribute has already been created using an **xattrschema create** operation described in “Extending Server Configurations” in this section.

```
dcecp> server modify /.:/hosts/silver/config/srvrconf/video_clip
                    -change {objfamily {c09dcc40-e4f4-11cd-bd59-0000c08adf56}}
dcecp>
```

Checking Whether Servers are Running

You can check whether a particular server is running by performing a **server ping** operation. This might be a convenient test when some client users report they cannot communicate with a server. The **server ping** operation communicates with the named server to test its presence, returning a **1** if a server is listening and a **0** if it is not listening.

Permissions Required — Controlling Server Operation

Permissions for Accessing Server Control Facilities

ACLs (access control lists) prevent unauthorized principals from creating, reading, changing, or deleting information maintained by the server control facilities.

The server control facility maintains two kinds of server control information. Server configuration information (named **srvrconf** in DCE) consists of the information needed to start servers. Server execution information (named **srvrexec** in DCE) consists of information needed to control or stop servers when they are running.

Server configuration information is protected by two types of ACLs. One ACL protects the container in which the server control information resides. A second ACL type protects each individual server's configuration information.

Similarly, server execution information is protected by two types of ACLs. One ACL protects the container in which the server execution information resides. A second ACL type protects each running server's execution information.

This section shows how to manage ACLs that protect server control information. For detailed information about setting and using ACL protections, read “Using Access Control Lists” on page 1-59.

Permissions for the Server Configuration Container

The server configuration information resides in a container. The container, a backing storage mechanism implemented as a file on AIX systems, is owned by root and is also protected by an ACL. These ACL permissions control who can access information in the container. Each DCE host has one server configuration container ACL with the following name:

/.../*cellname*/**hosts/***hostname*/**config/srvrconf**

The server configuration container ACL has the following permissions:

c (control) Modify the container ACL

r (read) Read configuration information in the container

i (insert) Create new configuration information

I (Insert) Create new configuration information for a server that runs as a privileged user (for example, as root on a POSIX system). Such operations also require the **i** permission.

Permissions for Accessing Server Configuration Information

Each server's configuration information is protected by its own ACL. These ACLs can prevent unauthorized principals from creating, reading, changing, or deleting server configuration information, and from starting, stopping, enabling and disabling servers.

Each ACL is named for the server configuration information it protects and has a name like:

/.../*cellname*/**hosts/***hostname*/**config/srvrconf/***server_name*

This ACL has the following permissions:

c (control) Modify the ACL

d (delete) Delete the server configuration information

f (flag) Start server with custom flags

r (read) Read the server configuration information

w (write) Modify the server configuration information

x (execute) Start server

Permissions for the Server Execution Container

When servers are started, the DCE daemon copies server configuration information into the server execution database. The **dced** process also adds more information about the running server such as a UUID, the server's communication endpoints and its process name and ID. The execution information controls the running server; for instance, the process ID is used to stop a server. When a server exits, the DCE daemon removes its server execution information.

The server execution information resides in a container. The container, a backing storage mechanism implemented as a file on AIX systems, is owned by root and its access through **dced** is protected by an ACL. These ACL permissions control who can access information in the container. Each DCE host has one server execution container ACL with the following name:

/.../*cellname*/**hosts/***hostname*/**config/srvrexec**

The server execution container ACL has the following permissions:

c (control) Modify the container ACL

r (read) Read execution information in the container

i (insert) Create new execution information

l (Insert) Create new execution information for a server that runs as a privileged user (for example, as root). Such operations also require the **i** permission.

Permissions for Accessing Server Execution Information

Each server's configuration information is protected by its own ACL. These ACLs can prevent unauthorized principals from creating, changing, reading, or deleting server configuration information, and from starting, stopping, enabling and disabling servers.

Each ACL is named for the server execution information it protects and has a name like:

/.../*cellname***/hosts/***hostname***/config/srvrexec/***server_name*

This ACL has the following permissions:

c (control) Modify the ACL

r (read) Read server execution information

w (write) Modify the server execution information

s (stop) Stop server

Examples — Controlling Server Operation

- The following example shows configuration information for the **video_clip** server on host **krypton** in remote cell **/.../their_cell.goodco.com**.

```
dcecp> server show /.../their_cell.goodco.com/hosts/krypton/config/srvrconf/video_clip
{uuid 2fa417e8-bb4c-11cd-831b-0000c08adf56}
{program {vclip}}
{arguments {-catalog}}
⋮
dcecp>
```

- The next example shows configuration information for the **video_clip** server on host **silver** in the local cell.

```
dcecp> server show /.:/hosts/silver/config/srvrconf/video_clip
{uuid 2fa417e8-bb4c-11cd-831b-0000c08adf56}
{program {vclip}}
{arguments {-catalog}}
⋮
dcecp>
```

- Use the simple name to show configuration information for the **video_clip** server on the local host:

```
dcecp> server show video_clip
{uuid 2fa417e8-bb4c-11cd-831b-0000c08adf56}
{program {vclip}}
{arguments {-catalog}}
⋮
dcecp>
```

- Use the **dcecp acl** object to view or modify ACLs. For example, use the following operation to view the ACL for the server configuration container object on host **silver**.

```
dcecp> acl show /.:/hosts/silver/config/srvrconf
{user appl_admin criI}
{unauthenticated r}
{any_other r}
dcecp>
```

Because /**.:/hosts**/**silver**/**config**/**srvrconf** is a container, it also has an Initial Container ACL and an Initial Object ACL. You can operate on these initial ACLs by using the **–ic** and **–io** options to **acl** operations. Note that the Initial Container ACL has no effect because currently, you cannot create child containers under /**.:/hosts**/*hostname*/**config**/**srvrconf**.

- Use the **acl** object to view or modify ACLs. For example, use the following operation to view the ACL for the **video_clip** server on host **silver**.

```
dcecp> acl show /.:/hosts/silver/config/srvrconf/video_clip
{user appl_admin cdfrwx}
{unauthenticated r}
{any_other r}
dcecp>
```

This ACL takes its default values from the container's Initial Object ACL. You can operate on the Initial Object ACL by using the **–io** option to **acl** operations. The following example shows the Initial Object ACL for the **video_clip** server:

```
dcecp> acl show /.:/hosts/silver/config/srvrconf -io
{unauthenticated r}
{any_other r}
dcecp>
```

- Use the **acl** object to view or modify ACLs. For example, use the following operation to view the ACL for the server execution container object on host **silver**.

```
dcecp> acl show /.:/hosts/silver/config/srvrexec
{user appl_admin criI}
{unauthenticated r}
{any_other r}
dcecp>
```

Because /**.:/hosts**/**silver**/**config**/**srvrexec** is a container, it also has an Initial Container ACL and an Initial Object ACL. You can operate on these initial ACLs by using the **–ic** and **–io** options to **acl** operations. Note that the Initial Container ACL has no effect because currently, child containers do not exist under /**.:/hosts**/*hostname*/**config**/**srvrexec**.

- Use the following operation to view the ACL for the server execution information for the **video_clip** server on host **silver**.

```
dcecp> acl show /.:/hosts/silver/config/srvrexec/video_clip
{user appl_admin crws}
{unauthenticated r}
{any_other r}
dcecp>
```

- This ACL takes its default values from the container's Initial Object ACL. You can operate on the Initial Object ACL by using the **–io** option to **acl** operations. The following example shows the Initial Object ACL for the **video_clip** server:

```
dcecp> acl show /.:/hosts/silver/config/srvrexec -io
{unauthenticated r}
{any_other r}
dcecp>
```

- The following example creates an entry for a fictitious video clip server named **video_clip** on the local host. For a remote host or a host in another cell, use the cell-relative or the global name. The program name **vclip** invokes the server which is located in the /**usr**/**local**/**bin** working directory. The server has a catalog mode that we set by specifying **–catalog** as the argument. The server uses DCE security so the server has a principal name **Vclip_Srv_1**. The **–entryname** option specifies the entry

name in CDS where the server stores its binding information. The **–starton** option sets the server to start when the **dced** receives an explicit **server start** operation that names the **video_clip** server. The **failure** attribute further specifies to restart the server if it exits with a status that is not successful. The **–services** option has annotation information to help administrators identify servers when this information is returned with **server show** operations. The **interface** attribute is needed because the DCE daemon copies this information into the host endpoint map when the server starts.

```
dcecp> server create /.:/hosts/silver/config/srvrconf/video_clip
                -program {/usr/local/bin/vclip}
                -directory {/tmp} -arguments {-catalog}
                -principal {Vclip_Srv_1}
                -entryname {/.:/subsys/applications/video_clip_1}
                -starton {explicit failure}
                -services {{annotation {Video Clip Catalog and Server}}
                     {interface {d860322b-d499-11cd-9dfb-0000c08adf56 1.0}}}
dcecp>
```

- The next example configures the same server to start whenever the host system boots. The only difference from the preceding example is that the **–starton** option has a value of **boot**.

```
dcecp> server create /.:/hosts/silver/config/srvrconf/video_clip
                -program {/usr/local/bin/vclip}
                -directory {/tmp} -arguments {-catalog}
                -principal {Vclip_Srv_1}
                -entryname {/.:/subsys/applications/video_clip_1}
                -starton {boot failure}
                -services {{annotation {Video Clip Catalog and Server}}
                {interface {d860322b-d499-11cd-9dfb-0000c08adf56 1.0}}}
dcecp>
```

- The final configuration example sets the **video_clip** server to start whenever a client request for its services is received at the host system. The **–starton** option value is **auto**.

```
dcecp> server create /.:/hosts/silver/config/srvrconf/video_clip
                -program {/usr/local/bin/vclip}
                -directory {/tmp} -arguments {-catalog}
                -principal {Vclip_Srv_1}
                -entryname {/.:/subsys/applications/video_clip_1}
                -starton {boot failure}
                -services {{annotation {Video Clip Catalog and Server}}
                {interface {d860322b-d499-11cd-9dfb-0000c08adf56 1.0}}}
dcecp>
```

- The following example removes the **video_clip** server's configuration information from the configuration database on host **silver**.

```
dcecp> server delete /.:/hosts/silver/config/srvrconf/video_clip
dcecp>
```

- The following **server start** operation starts the explicit server **video_clip** on host **silver** in the local cell.

```
dcecp> server start /.:/hosts/silver/config/srvrconf/video_clip
dcecp>
```

- The next example stops the explicit server **video_clip** on the local host **silver** in the local cell.

```
dcecp> server stop video_clip
dcecp>
```

- The following example disables one service of the **video_clip** server.

```
dcecp> server disable /.:/hosts/silver/config/srvrexec/video_clip
                -interface {d860322b-d499-11cd-9dfb-0000c08adf56 1.0}
dcecp>
```

- The next example enables the **vidsrv** service of the **video_clip** server after it has been disabled. This operation allows clients to acquire a server's endpoint.

```
dcecp> server enable /.:/hosts/silver/config/srvrexec/video_clip
               -interface {d860322b-d499-11cd-9dfb-0000c08adf56 1.0}
dcecp>
```

- You can disable or enable all services offered by a server by using a **server disable** or **server enable** command with no options. The following example disables all services offered by **bbs_server** on host **krypton**:

```
dcecp> server disable /.:/hosts/krypton/config/srvrexec/bbs_server
dcecp>
```

- If you have a server that needs an attribute that specifies an object family. You create such an attribute using the **xattrschema** object. The following example creates an ERA called **objfamily**. The operation specifies the permissions needed to query, update, test, and delete the ERA, and it specifies the ACL manager that supports the permissions.

```
dcecp> xattrschema create /.:/hosts/silver/config/xattrschema/srvrconf/objfamily
                    -attribute {{annotation {object family}} {encoding uuid}
                    {aclmgr {srvrconf r w r d}
dcecp>
```

- You can review the attributes associated with an ERA by using an **xattrschema show** operation as shown in the following example:

```
dcecp> xattrschema show /.:/hosts/silver/config/xattrschema/srvrconf/objfamily
{aclmgr {srvrconf {{query r} {update w} {test r} {delete d}}}}
{annotation {object family}}
{applydefs no}
{encoding uuid}
{intercell reject}
{multivalued yes}
{reserved no}
{scope {}}
{trigbind {}}
{trigtype none}
{unique no}
{uuid 1bef2222-e687-11cd-b74a-0000c08adf56}
dcecp>
```

- Use the **dcecp acl** object to view or modify the container ACL. For example, the following operation views the ERA container ACL on host **silver**.

```
dcecp> acl show /.:/hosts/silver/config/xattrschema
{user appl_admin criI}
{unauthenticated r}
{any_other r}
dcecp>
```

- In this example, the **c** permission has no effect because it was not assigned when the ERA was created with the **xattrschema create** operation. All users can query and test the ERA. Only the user named **appl_admin** can also update and delete the ERA.

```
dcecp> acl show /.:/hosts/silver/config/xattrschema/srvrconf/objfamily
{user appl_admin crwd}
{unauthenticated cr}
{any_other cr}
dcecp>
```

- This ACL takes its default values from the container's Initial Object ACL. You can operate on the Initial Object ACL by using the **–io** option to **acl** operations. The following example shows the Initial Object ACL for the **xattrschema** container on host **silver**:

```
dcecp> acl show /.:/hosts/silver/config/xattrschema -io
{unauthenticated cr}
{any_other cr}
dcecp>
```

- The following example tests whether the **video_clip** server is running.

```
dcecp> server ping /.:/hosts/silver/config/srvrconf/video_clip
1
dcecp>
```

DCE Host Services

Introduction — DCE Host Services

Some DCE host services, such as the runtime libraries, are inert and require no administration after DCE has been configured on a host. But other services are active programs. One such active service is the endpoint mapper which acts as a lookup service on a host. The endpoint mapper lists server communication ports (called *endpoints*) in the host's endpoint map. Remote clients looking for particular servers query the endpoint mapper which returns information contained in the endpoint map. The endpoint mapper, along with other active services, are contained in a single program called the DCE Host Daemon or **dced**. Typically, after a host has been configured with DCE for AIX Version 2.1 software, the host booting process starts the **dced** process along with other daemons or processes. Occasionally however, you may need to manually start or restart this daemon.

More Detailed Information — DCE Host Services

The **dced** program comprises a set of DCE host services that satisfies many needs of DCE client and server applications on a host system:

- The endpoint mapper service acts as a directory of servers running on a host. Clients can acquire a registered server's communication endpoint by looking in the host endpoint map.
- A security validation service manages DCE security on the local host.
- A server configuration and execution service lets administrators remotely set servers' starting and stopping conditions, explicitly start and stop individual servers, and monitor running servers' states.
- A key management service lets administrators manage server passwords remotely.
- A host data service lets administrators remotely manage data stored in files on a host. Administrators will find this most useful for remotely managing a host's cell name and cell alias information.
- An attribute schema capability lets administrators add new attributes to server configuration information.

Normally, any system that hosts a DCE server (such as a DCE Cell Directory Server) or that runs a DCE-based application server or client which uses authentication, must also run the **dced** program.

It is obvious that if the **dced** program failed for some reason, it would take all of its component services down along with it, leaving the host unable to respond to client requests. Similarly, a failure of one of the component services (for example the key management service) might be caused by the **dced** program unexpectedly exiting for some reason. This relationship between **dced** and its component services is worth remembering if problems occur.

Starting and Stopping DCE Host Services

Although the **dced** program generally starts as part of the host booting process, sometimes you may need to start the program manually.

Before starting **dced**, any underlying network services on which client/server communication depends must be available; on most UNIX systems, for example, network interfaces and routing services must be

enabled. After establishing these transport-layer services, you can start **dced**. After **dced** starts, RPC-based servers can start.

The endpoint mapper listens on privileged or reserved communication ports (well-known endpoints) for client requests for service. Consequently, **dced** must be started as a privileged user.

Part of **dced** (the endpoint map) contains information that clients use to locate servers on a host system. The **dced** program maintains a copy of this information in a database file named **/opt/dcelocal/var/dced/Ep.db** so it will not be lost if you stop and then restart **dced** for some reason. Another database file called **/opt/dcelocal/var/dced/Srvrexec.db** maintains information about servers (such as each server's process id) that are currently running on the host. The information in both of these databases becomes obsolete when a system reboots because most servers get different endpoints and different process IDs each time they start.

You can configure **dced** to start each time a host boots. In some cases, the **/opt/dcelocal/etc/rc.dce** file is linked to or copied to **/etc/rc.dce**. This way, **dced** is invoked when other daemons are started. The **/opt/dcelocal/etc/rc.dce** file is also responsible for deleting the **Srvrexec.db** and **Ep.db** database files before starting **dced**. If your system does not automatically delete these files, you will have to manually delete them before starting **dced**.

While you normally do not need to start **dced** in a shell, if you ever need to do so, become the superuser and enter the following command:

```
$ /opt/dcelocal/bin/dced
```

By default, **dced** listens on one endpoint for each transport that is supported by the host on which it is running. That is, if a host supports both TCP/IP and UDP/IP transports, **dced** will listen on one TCP and one UDP socket for client requests. An optional *protseq* argument lets you limit the transports that **dced** uses to the ones you specify. Intended as a debugging capability, this feature should be used with care; if you limit transports, clients will not be able to locate servers over the excluded transports, and servers will not be able to register themselves in the endpoint map by using the excluded transports. For information about the optional *protseq* argument, see the **dced** reference page in the *DCE for AIX Administration Command Reference*.

If the DCE daemon stops or exits unexpectedly, you can restart it. The restarted **dced** does *not* lose any previously registered server bindings. It simply loads the information from the **Ep.db** and **Srvrexec.db** files. As a rule, stopping and restarting the **dced** is not recommended because it also stops the security validation service.

Although you should run the host services on all hosts where DCE client or server applications run, there are some situations where you can avoid running them:

- DCE clients that do not perform authentication
- DCE servers that do not perform authentication and that don't use the endpoint mapper or other active DCE host services

After starting the DCE host services, you can perform all DCE host and server administration tasks by using the DCE Control Program (**dcecp**). The control program offers secure, remote access to host and server administrative functions which means you can manage all of your DCE for AIX Version 2.1 hosts without having to log into each host. In other sections of this guide we explain how to use **dcecp** in interactive mode as well as how to write **dcecp** scripts to manage DCE activities. It will help to understand those basics before performing administrative tasks explained in this guide.

Easing Application Development and Administration

Introduction — Easing Application Handling

Remote clients can find a server by using the server host's endpoint map to determine the server's communication endpoint. But how do remote clients know where to find the endpoint map itself? They know because the endpoint map is always accessible at a well-known endpoint (that is, it is always the same endpoint) on each host so clients can easily find it.

When hosts support multiple transports, the endpoint map listens on one port for each transport. In the IP address family (both TCP and UDP), the endpoint map process listens on port 135. In the Domain Domain Sockets (DDS) address family, it listens on port 12. In the DECnet NSP address family, it listens on port 69. A complete list of the protocol sequences and well-known endpoints used by the Endpoint Map Service can be found in the header file **/opt/dcelocal/share/include/dce/ep.idl**. Note that not all hosts support all transports. DCE software tries to ensure that at least one transport is shared between a client and a server.

More Detailed Information — Easing Application Handling

While well-known endpoints provide convenient access to some critical servers, for most servers they are impractical. That is because some address families have a limited number of endpoints and well-known endpoints can be assigned only by a central administrative authority. So most servers use dynamic endpoints. When a server starts up, the RPC runtime library gets an available endpoint from the operating system and registers it in the host endpoint map.

Because a server can be assigned a different endpoint each time it starts, the endpoint information is stored in the endpoint map rather than CDS, which is a repository for more stable information; namely, the server's host address and the transports it uses. As long as the server stays on the same machine, host and transport information need not be updated, which tends to reduce bottlenecks at CDS.

This scheme makes application development and administration easier because it reduces the need to manage endpoints. Servers need not worry about passing dynamic endpoints to clients. Furthermore, unless a server moves to a new host, or removes or adds a transport, it does not even have to update the information in CDS.

Endpoint Map Administration is Mostly Automatic

Each server that uses the endpoint map stores a set of information in the endpoint map when it starts up. The information includes UUIDs (Universal Unique Identifiers) for objects and interfaces offered by the server, an annotation string, and other fields.

The endpoint map resides on disk in **/opt/dcelocal/var/dced/Ep.db** and **/opt/dcelocal/var/dced/Srvrexec.db**. After a system reboot, DCE-based servers restart and reregister with the Endpoint Map Service, so the database files need to be deleted before the DCE daemon starts. This happens automatically on most systems.

DCE-based servers normally need to register with the Endpoint Map Service on startup and unregister on termination. If any servers exit without unregistering, the endpoint map may contain stale entries. DCE for AIX Version 2.1 provides server control facilities that help servers unregister and avoid leaving stale entries in the endpoint map. Servers that do not use these facilities (older servers, for example) are more likely to leave stale entries if they exit unexpectedly. So periodically, the DCE daemon (*dced*) purges stale entries by scanning the endpoint map, pinging each server that is registered, and deleting entries for servers that do not respond.

The background process of removing stale entries is not intended to be highly responsive. It is not intended to replace the need for servers to unregister themselves from the endpoint map when they no longer service RPCs. Rather, this processing is intended only to clean up after a server failure.

While the behavior of the pinging/purging mechanism is implementation dependent, in a typical implementation the database is scanned (that is, servers are pinged and stale entries removed) only infrequently; for example, a few times an hour. When a ping to a server fails, the server is pinged several times over a shorter interval (for example, every 5 minutes). If the server continues to not respond, the *dced* determines that its entry is stale and removes it from the database. Ultimately, the rate at which stale server entries are detected and purged depends on the number of stale entries in the database; the more stale entries, the longer it takes to detect and purge the stale entries.

Restricting Endpoints

You can restrict the assignment of endpoints (ports) for DCE servers and clients to a specific set. This is useful if your environment has non-DCE applications that are designed to use certain endpoints, and you do not want to be concerned about DCE servers or clients monopolizing them.

The facility is activated by setting the **RPC_RESTRICTED_PORTS** environment variable with the list of endpoints to which dynamic assignment should be restricted before starting a client or server application. **RPC_RESTRICTED_PORTS** governs only the dynamic assignment of server ports by the RPC runtime. It does not affect well-known endpoints.

Note that this facility does not add any security to RPC and is not intended as a security feature. It merely facilitates configuring a network firewall to allow incoming calls to DCE servers.

Displaying Information in the Endpoint Map

For the most part, the endpoint map on each host takes care of itself, purging stale entries when necessary and removing the endpoint information each time the host reboots. So there is really no administration needed for the endpoint map.

However, when client/server communication problems arise, the information stored in the endpoint map might be useful to administrators, particularly for determining whether servers are supplying the correct endpoint information to clients. In this case, you can use the **endpoint** object to view endpoint map information. Besides its use in troubleshooting, you can also use the **endpoint** object for other specialized server operations such as adding new object UUIDs to existing mappings.

Endpoints are not protected by ACLs. This means anyone who can run *dcecp* can use an **endpoint show** operation on their host to view endpoint information on any other host in the cell. Other endpoint operations such as creating or deleting endpoints can be performed only by users that are logged into the local host. No other special privileges, such as system administrator or root privileges, are needed for local access to endpoint information.

You can view information stored in a host's endpoint map database by using an **endpoint show** operation.

You can view all of the endpoints in an endpoint map by not using any options with the **endpoint show** operation.

Examples — Easing Application Handling

- The following example restricts servers to using TCP/IP endpoints ranging from 5000 to 5110, and 5500 to 5521. It restricts UDP/IP endpoints to the range of 6500 to 7000.

```
% set RPC_RESTRICTED_PORTS ncacn_ip_tcp[5000-5110,5500-5521]:ncadg_ip_udp[6500-7000]
%
```

To use **RPC_RESTRICTED_PORTS** for DCE servers such as CDS, set the environment variable each time before starting your cell.

- The following example shows the endpoint map information for the **video_clip** server on a remote host megazoid. Omit the *hostname* argument to operate on the local endpoint map.

```
dcecp> endpoint show ncacn_ip_dcp:130.105.1.227
                     -interface {2fa417e8-bb4c-11cd-831b-0000c08adf56 1.0}
{{object 99ff4fb8-c042-11cd-91cd-0000c08adf56}
 {interface {2fa417e8-bb4c-11cd-831b-0000c08adf56 1.0}}
 {binding {ncacn_ip_tcp 130.105.1.227 1028}}
 {annotation {Text Development Utilities}}}
dcecp>
```

Managing Client/Server Binding Information

Introduction — Managing Client/Server Binding Information

In a DCE environment, clients and their servers frequently reside on different hosts in a network so clients need a way to find servers.

More Detailed Information — Managing Client/Server Binding Information

Clients need three pieces of information to communicate with a server:

- The DCE host name (or network address) of the host where the server is running
- The name of the network transport the server is using
- The communication port (endpoint) the server is using for client communications

Of course, an application programmer could simply hard-code a server's location information (also called binding information) into the client side of the application where it is immediately available for use. However, this approach requires that a programmer have advance knowledge of precise network details such as DCE host names and available port numbers. Furthermore, servers with hard-coded binding information do not easily adapt to configuration changes. If you move a server to a different host, you need to recompile all of the clients with the server's new DCE host name. So DCE provides more flexible ways for clients to obtain server bindings.

The standard way for clients to find servers is by using CDS and the server host's endpoint map. Figure 4-1 on page 4-18 provides a high-level example of this method showing how a fictitious dictionary client application on host **larry** finds a dictionary server on host **curly**.

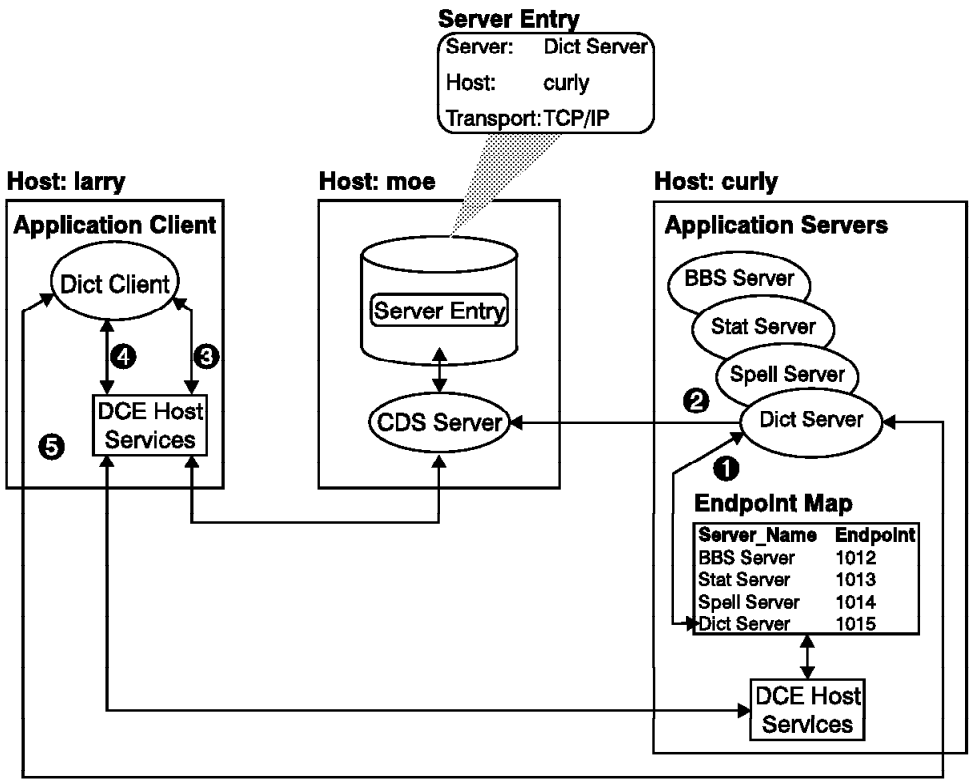


Figure 4-1. Server Binding Information

1. When the dictionary server starts up, DCE host software assigns the server a communications port (endpoint), which clients will use to communicate with this server. Here, the endpoint is TCP/IP port 1015. The DCE host software also places the server identification information along with the current endpoint in the host's endpoint map.
2. The dictionary server then advertises its availability to clients by placing (exporting) its DCE host name (usually it is the host address) and the transport it uses to a server entry in CDS.
3. When the dictionary client makes a call to a remote procedure provided by the server, the DCE software on the client queries the CDS server to find the dictionary server's DCE host name and the transport.
4. The client system's host software then queries the endpoint map on host **curly** to find the dictionary server's endpoint (port 1015).
5. Equipped with all the necessary binding information, the host services on host **larry** transmit the remote procedure call directly to port 1015 on host **curly**.

Although we have omitted some details in this high-level example, the figure still shows the major binding activities performed by clients and servers. That is, servers place their binding information in CDS and in the host endpoint map where clients look for it. There are other ways for clients to find servers and there are variations on the mechanism we have described. But these alternatives are generally controlled by the applications themselves rather than through conventional DCE administration facilities like **dcecp**.

Managing Host Data

Introduction — Managing Host Data

Each host in a DCE cell maintains local data that is essential for operating in a DCE environment. For example, the DCE identity of each host relies on certain data items that specify the DCE host name, cell name, and any cell aliases of the host. Currently, these data items are stored in a local file called **/opt/dcelocal/dce_cf.db**. These and other data items can be modified remotely using the DCE control program's **hostdata** object.

The **hostdata** object has a much broader application, too; administrators will find it extremely useful for accessing general data and files on remote hosts using secure and platform-independent methods.

More Detailed Information — Managing Host Data

Permissions for Accessing Host Data

Access control lists (ACLs) prevent unauthorized principals from creating, changing, or deleting host data information. Two types of ACLs protect host data information. One type of ACL protects the container in which the host data items reside. A second type protects each individual host data item.

This section shows how to manage ACLs that protect host data information. For detailed information about setting and using ACL protections, read “Using Access Control Lists” on page 1-59.

Permissions for the Host Data Container

In DCE for AIX Version 2.1, the host data items reside in a container that is really a backing storage mechanism maintained by **dced**. On AIX systems this is usually a file called **/opt/dcelocal/var/dced/Hostdata.db**. The file is owned by root and its access through **dced** is protected by an ACL. These ACL permissions control who can access the data in the container. Each DCE host has one host data container ACL with the following name:

/.../*cellname***/hosts/***hostname***/config/hostdata**

The host data container ACL has the following permissions:

c (control) Modify the container ACL

r (read) Read the list of host data items in the container

i (insert) Create new host data items

I (Insert) Although the **I** permission is present, it does not apply to host data items. The permission applies to server control facilities which are explained in “DCE Application Administration” on page 1-27.

Permissions for the Host Data Items

Each of the following host identity data items is protected by an ACL:

/.../*cellname***/hosts/***hostname***/config/hostdata/host_name**

/.../*cellname***/hosts/***hostname***/config/hostdata/cell_name**

/.../*cellname***/hosts/***hostname***/config/hostdata/cell_aliases**

/.../*cellname***/hosts/***hostname***/config/hostdata/post_processors**

Each ACL has the following permissions:

c (control) Modify the ACL

d (delete) Delete the item

p (purge) Delete the backing storage for an item

r (read) Read an item's data

w (write) Modify an item's data

Modifying Host Cell Name Information

Using the **hostdata** object, you can add, change, and remove data items on DCE hosts. While administrators will find this useful for modifying a host's cell name or cell alias information, they can also operate on other data that is accessible on a host.

Each DCE host maintains a protected local copy of the cell name and cell aliases of the cell in which the host is registered. Hosts keep this information in a local file called /**opt**/**dcelocal**/**dce_cf.db** which is owned by **root**. Hosts use this information for authentication purposes — as part of its host identity information.

Although host cell name information tends to be fairly stable, there are circumstances where it is necessary to change this information, such as:

- When a host moves to a different cell, or
- When the host cell name changes or the cell name acquires an alias.

When either of these situations occurs however, it is usually not enough to just update the cell name information on the host. Cell name information must also be updated in CDS and in the DCE security service registry as well.

Sometimes however, the higher-level **dcecp** task objects do not offer enough control such as you might need when fixing a corrupted file somewhere or when configuring a host by hand for some reason. In these cases, you can use the **hostdata** object to change cell name information on individual hosts.

Note though, that this use of the **hostdata** object is intended mostly to be a troubleshooting operation to be relied on when a host's cell information is out of synchronization with other cell information stored in the DCE registry or stored in CDS. This situation might be a common occurrence in cells with many hosts.

Manipulating Data in Other Host Files

While the **hostdata** object is useful for changing cell name and cell alias information, it has a broader use too; you can use it to add, change, and remove data from any file that is accessible on a DCE host.

Examples — Managing Host Data

- Use the **acl** object in **dcecp** to view or modify ACLs. For example, use the following operation to view the ACL for the **hostdata** container object on host **silver**.

```
dcecp> acl show /.:/hosts/silver/config/hostdata
{user hosts/silver/self criI}
{unauthenticated r}
{any_other r}
dcecp>
```

- Use the **acl** object to view or modify ACLs. For example, use the following operation to view the ACL for the **cell_aliases hostdata** object on host **silver**.

```
dcecp> acl show /.:/hosts/silver/config/hostdata/cell_aliases
{user hosts/silver/self cdprw}
{unauthenticated r}
{any_other r}
dcecp>
```

- To update the cell name or cell alias name information on a host, use the **hostdata** object. The following example catalogs the **hostdata** objects in the cell named /**.../my_cell.goodco.com**. Then it shows the contents of the **cell_name** object on host **silver**. Finally it modifies the cell name to be /**.../my_cell.goodco.com** on host **silver**.

```
dcecp> hostdata cat
/.../my_cell.goodco.com/bronze/config/hostdata/dce_cf.db
/.../my_cell.goodco.com/bronze/config/hostdata/cell_name
/.../my_cell.goodco.com/bronze/config/hostdata/host_name
/.../my_cell.goodco.com/bronze/config/hostdata/cell_aliases
/.../my_cell.goodco.com/bronze/config/hostdata/post_processors
dcecp> hostdata show cell_name
{uuid 00174f6c-6eca-1d6a-bf90-0000c09ce054}
{annotation  {Name of cell}}
{storage cell_name}
{data {/.../old_cell.goodco.com}}
dcecp> hostdata modify /.../my_cell.goodco.com/hosts/bronze/config/hostdata/cell_name
                    -data {/.../my_cell.goodco.com}}
dcecp>
```

- One useful example is adding a new CDS attribute. Every DCE host has it's own CDS attributes file (**cds_attributes**) where it stores object IDs for each CDS attribute. You could use the local host's editor to add the attribute and then copy the new file to each host. But this method requires you to log in to each host. A simpler method could use the **hostdata** object to add the new attribute to the CDS_attributes file. Place the operation within a **foreach** loop that re-executes it for each host in the cell.

  1. Make the CDS attributes file accessible as an object of the **hostdata** object. First, use the **hostdata** object to create a CDS entry representing the CDS attributes file. Set the storage attribute to be the host file name of the CDS attributes file The following example assumes the CDS attributes file is in the default location.

     ```
     dcecp> hostdata create /.:/hosts/silver/config/hostdata/cds_attr
                     -storage /opt/dcelocal/etc/cds_attributes -entry
     dcecp>
     ```

  2. The **hostdata** object modifies data in files by replacing all the data in the file with new data that you specify. The following example shows one way to do this. First, retrieve and store all the lines as **dcecp** list elements in a variable. Then create a new variable using the **attrlist** command to add the new line as a list element to the variable. Finally, copy the new variable back to the file.

     ```
     dcecp> set val [hostdata show /.:/hosts/silver/config/hostdata/cds_attr]
     dcecp> set newval [attrlist add $val -member {NEW_ATTR 1.2.3.4}]
     dcecp> hostdata modify /.:/hosts/silver/config/hostdata/cds_attr -data $newval
     dcecp>
     ```

Managing Server Entries, Groups, and Profiles in CDS

Introduction — Managing Entries, Groups, and Profiles in CDS

This section explains how to use CDS facilities for organizing your servers and other distributed objects in meaningful ways.

Many of the operations discussed here operate on CDS directories which are protected against unauthorized access by ACLs (access control lists). For detailed information about ACLs and CDS see "Controlling Access to CDS Names" on page 1-29.

More Detailed Information — Managing Entries, Groups, and Profiles in CDS

Unique Server Entry Names Identify Individual Servers and Objects

We know that servers store their binding information in CDS where clients can find it. But so far, we have been treating CDS like a black box. If a DCE cell consisted of just a few servers or objects and a handful of users, CDS could be as simple as a data file assessable to both servers and clients. Finding unique names for objects would probably not pose a big problem. And you could probably even devise some effective scheme for protecting objects from unauthorized use. But DCE cells can include many hundreds or even thousands of objects. Large cells will likely contain many similar or even identical servers which need convenient and effective ways to offer their services to clients.

DCE CDS answers this need by providing a hierarchical (tree-structured) name system that servers use to store binding information. CDS acts much like a hierarchical file system of directories that stores names and other information instead of files. You can build on its hierarchical structure, imposing directory names that can correspond to your company's organizational structure.

Servers have CDS names like /**.:/admin/finance/payroll/check_writer**. When this check_writer server exports its server entry name to CDS, CDS stores it in a directory named /**.:/admin/finance/payroll**. Consequently, clients will not confuse this check_writer with another check_writer named /**.:/admin/finance/accts_payable/check_writer**. Thus, unique server entry names fill a critical administration need, providing a way to access and control individual servers. Other sections of this book provide more information about CDS and the structure and uses of CDS names. For our current purposes, it is enough to know how and why CDS directory names help make potentially identical server entries unique.

While servers themselves often manage exporting and removing their names and binding information from CDS, sometimes administrators need to manually add, change, or remove binding information. For instance, when a server host machine crashes unexpectedly and stays offline for a long time, its resident servers cannot remove their entry names and binding information from CDS. Clients can waste time looking for these phantom servers. The **dcecp** program provides the **rpcentry** object that you can use to manage server entry names and their binding information in CDS.

Before we get to the actual management tasks, let us examine a server entry to see exactly what it is we will be managing. Figure 4-2 on page 4-23 shows possible information in a server entry.

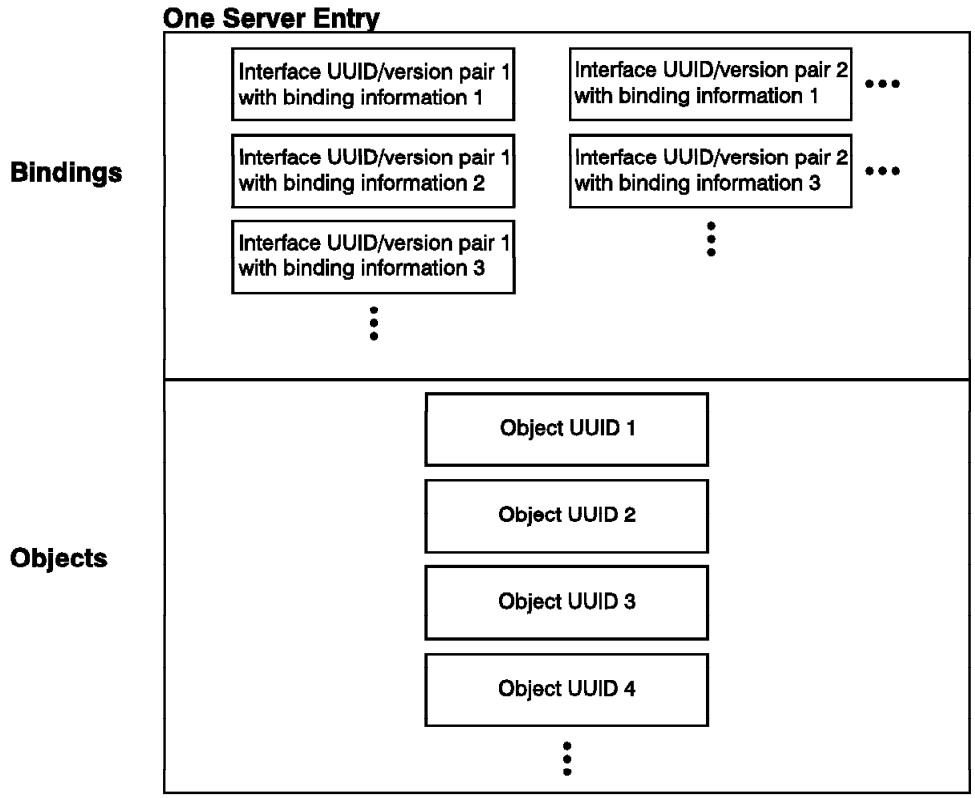


Figure 4-2. Possible Information in a Server Entry

The top part of the figure contains bindings. Each binding consists of an interface UUID/version pair (called an *interface identifier*) and a binding. The interface identifier identifies an interface offered by the server and its binding information indicates the host address and network transport to use to access that interface.

When an interface identifier is available over several transports, the server entry contains bindings (one binding for each transport). Servers can offer more than one interface. Multiple interfaces can be available through a single endpoint. That is, different interfaces can have the same bindings.

The lower part of the figure contains object UUIDs. Object UUIDs offer additional information to clients; they identify specific objects or resources managed by the server. For instance, one print server offers printers on floor 2 while another print server offers printers on floor 1. In this case, object UUIDs let clients select printers on the appropriate floor. In other words, object UUIDs help clients distinguish from among otherwise identical services.

Although application servers can manage their own server entries in CDS, you may find it more convenient (and more straightforward) to manually add, remove, or change information in a server entry. There are four methods for managing server entries in CDS:

- Server entry names can be hard coded into an application. You can change server entry information in the source code but you need to recompile and rerun the application before the entry names take effect.
- Server entry names can be stored as the **entryname** attribute of the server's configuration information (using the **server** object) where it is accessible to the application. This is more convenient than recompiling but, more importantly, this method places the server's entry name in a standard (platform-independent) place where administrators can see it too. You might need to restart an application to use this method, however.

- Server entry names can be passed to an application through environment variables or arguments. While these are effective methods and they are more convenient than recompiling, they are not platform-independent. This means you might need different approaches on different operating systems.
- Server entry names can be directly managed in CDS by using the DCE control program's **rpcserver** object. This manual method does not require recompiling or restarting applications.

The next sections discuss how to use the **rpcserver** object to manually manage server entries in CDS.

Creating a Server Entry in CDS

Often, servers will create their own entries in CDS either when they initialize or when they are configured after installation. But sometimes, you might want to create a server entry manually. When you create a server entry, it is empty; it does not contain any interface or binding information.

One reason to create an empty server entry is to establish ownership of the entry. Server entries are owned by the creator. If a server creates an entry, the server can also delete the entry later. You can preempt such a circumstance by creating the entry yourself. Later, the server exports its bindings to the existing server entry (provided the ACL (access control list) allows this).

Deleting a Server Entry from CDS

Because server entries generally contain stable server binding information, they tend to stay around rather than be deleted. Even when a server goes away for a short time, say, overnight, it might not be practical to remove its entry. But when a server goes away for a long time, you can avoid the client expense of trying to use the phantom server by removing the server's entry from CDS.

Exporting Binding Information to a Server Entry in CDS

Servers usually export their own binding information to CDS when they initialize or when they are configured after installation. But sometimes, binding information may have been removed for some reason or by accident and you want to restore it. Or another transport has been added and you want to export the binding for the new transport.

You can manually export server binding information to a server entry using an **rpcentry export** operation. If the entry does not already exist, the **rpcentry export** operation creates it provided the directory already exists and you have the necessary permissions.

Importing Binding Information from a Server Entry in CDS

Application client programs can automatically import server binding information from CDS and use it in their quest to find and communicate with a server. But occasionally, an administrator might want to import a binding. For instance, a client might lack access to CDS but it could still communicate with the server if you supplied it with a valid binding.

Viewing Information in a Server Entry

When clients are having difficulty communicating with servers, you might want to see what binding information is contained in a server entry as a troubleshooting step. Or say you are adding object UUIDs to server entries and you wonder whether a server entry has been overlooked.

Removing Binding Information from a Server Entry in CDS

Occasionally, you might want to remove binding information from a server entry. If a server host crashes, its servers cannot remove their server entries from CDS. To prevent clients from trying to communicate

with these phantom servers, you should unexport the bindings from CDS manually. Unlike the **endpoint delete** operation, this operation does not remove the entry name from CDS.

Group Entries Help Balance Server Workloads

When a client queries CDS for a server binding, the request includes the name of the entry to look in for the binding. When only one server offers the client's requested service, CDS will return the same binding for every client request for this service. While this model works fine for limited client requests, it can cause service bottlenecks when many client requests converge on one server. Applications can avoid bottlenecks by providing multiple servers to service large numbers of client requests. Server entry names alone do not provide a convenient way to distribute client requests evenly among multiple servers because you would have to explicitly direct each client to a particular server. So CDS provides **group entries** as a convenient mechanism for distributing the client load across multiple servers.

A CDS group entry gathers related servers together under a common group name. Group entries contain members which are generally pointers to server entries but members can point to other group entries, too. When a client requests a binding from a group entry, CDS returns, at random, one of the pointers contained in the group entry. If the entry picked at random is another group entry, CDS does not return that. Instead CDS goes to that group and picks another random member, continuing until a server entry is returned. This model requires that any group member can service the client request. Figure 4-3 shows how a group entry contains members that point to other groups and to server entries.

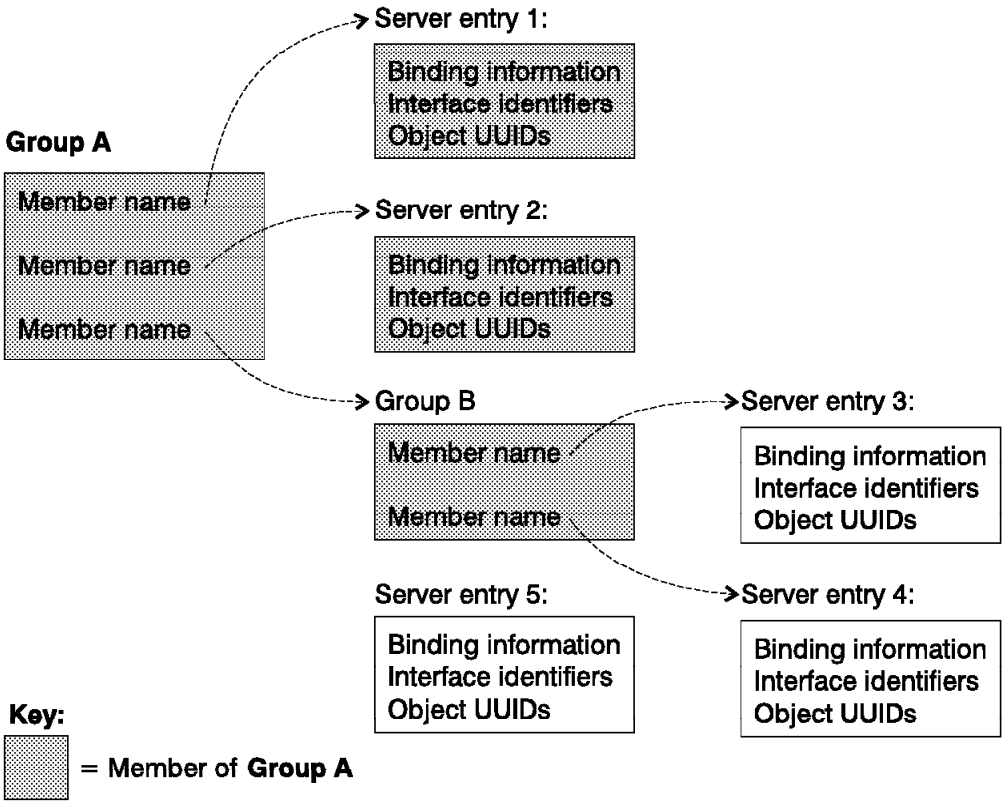


Figure 4-3. Possible Mappings of a Group

To see how group entries help balance a workload, consider an organization with twelve identical laser printers equally spread among three departments. The following group entry examples show how each group entry name returns any one of the four printers assigned to its own department.

```
Group entry name: /.:/admin/finance/accts_payable_printers
  /.:/admin/finance/accts_payable/laser_10
  /.:/admin/finance/accts_payable/laser_11
  /.:/admin/finance/accts_payable/laser_12
  /.:/admin/finance/accts_payable/laser_13

Group entry name: /.:/admin/finance/accts_receivable_printers
  /.:/admin/finance/accts_receivable/laser_10
  /.:/admin/finance/accts_receivable/laser_11
  /.:/admin/finance/accts_receivable/laser_12
  /.:/admin/finance/accts_receivable/laser_13

Group entry name: /.:/admin/finance/payroll_printers
 /.:/admin/finance/payroll/laser_10
 /.:/admin/finance/payroll/laser_11
 /.:/admin/finance/payroll/laser_12
 /.:/admin/finance/payroll/laser_13
```

You could temporarily make one department's printers available to another group by adding its group name to the group entry of the other group as shown in the next group entry example.

```
Group entry name: /.:/admin/finance/accts_payable_printers
  /.:/admin/finance/accts_payable/laser_10
  /.:/admin/finance/accts_payable/laser_11
  /.:/admin/finance/accts_payable/laser_12
  /.:/admin/finance/accts_payable/laser_13
  /.:/admin/finance/accts_receivable_printers
```

The configuration in the preceding example means the clients in accounts payable can use the printers in accounts receivable 20% of the time. You could offer a higher percentage of use by adding server entry names rather than the group name. The next group entry example shows a situation where the clients in accounts payable can use the printers in accounts receivable 50% of the time. One caveat; do not try to increase the percentage of use by including a group name multiple times because you will get an error.

```
Group entry name: /.:/admin/finance/accts_payable_printers
  /.:/admin/finance/accts_payable/laser_10
  /.:/admin/finance/accts_payable/laser_11
  /.:/admin/finance/accts_payable/laser_12
  /.:/admin/finance/accts_payable/laser_13
  /.:/admin/finance/accts_receivable/laser_10
  /.:/admin/finance/accts_receivable/laser_11
  /.:/admin/finance/accts_receivable/laser_12
  /.:/admin/finance/accts_receivable/laser_13
```

Although application servers can manage their own group entries in CDS, you may find it more convenient (and more straightforward) to manually add, remove, or change server information in a group entry. Like managing server entries, there are several methods for managing group entries in CDS:

- Group entry names can be hard coded into an application. You can change group entry information in the source code but you need to recompile and rerun the application before the entry names take effect.
- Group entry names can be passed to an application through environment variables or arguments. These are more convenient methods than recompiling but you might need to restart an application to use either method.
- Group entry names can be directly managed in CDS by using the DCE control program's **rpcgroup** object. This manual method does not require recompiling or restarting applications.

The next sections discuss how to use the **rpcgroup** object to manually manage group entries in CDS.

Creating a New Group Entry in CDS

You can create an empty group entry in CDS by using an **rpcgroup create** operation. While group creation is frequently performed by applications that first use a group entry, creating an entry yourself establishes you as the owner of the entry. As the owner, you have ultimate control over who can export and manage information in the entry.

Adding a Member to a Group Entry in CDS

You can use an **rpcgroup add** operation to add a member to a group entry. If the group entry does not exist, the operation creates the group entry and adds the member. The member can be a server entry or another group entry. Note that no operations check whether the members you add actually exist. This lets you configure the namespace even before servers are up and running.

Viewing the Members of a Group Entry

You can list the members of a group entry by using an **rpcgroup list** operation. This is useful for troubleshooting or for just seeing how servers are distributed in group entries.

Importing Binding Information from a Group Entry in CDS

Application client programs can automatically import server binding information from CDS and use it in their quest to find and communicate with a server. But occasionally, an administrator might want to import a binding. In the case where a client lacks access to CDS, it could still communicate with the server if you supplied the client with a valid binding.

You can use other options such as **–version** and **–object** to further specify a binding. Use the **–max** option to limit the number of bindings returned.

Removing Members from a Group Entry in CDS

Over time, organizational changes can require you to redeploy servers in your DCE cell. You might, for instance, want to move server entries from one group entry into another.

Deleting a Group Entry from CDS

Organization changes or server redeployments can make some groups obsolete. When you want to remove a group entry from CDS, use an **rpcgroup delete** operation.

Using Profiles to Direct Client Searches for Servers

Group entries offer clients a random choice from among multiple available services. Although a group entry can help in load balancing and resource allocation, its random nature resists fine tuning. Furthermore, it does not offer a way to prioritize servers for use by particular clients.

Profiles offer a complimentary way to organize servers because you can prioritize the search order of the profile members. (These were called elements in previous DCE versions.) Members identify servers by providing the following information:

- **Interface Identifier** — This field is the key to the profile. The interface identifier consists of the interface UUID and the interface version numbers.
- **Member Name** — The entry name of one of the following kinds of directory service entries:
  - A server entry for a server offering the requested RPC interface
  - A group corresponding to the requested RPC interface
  - A profile

- **Priority Value** — The priority value (0 is the highest priority; 7 is the lowest priority) is designated by the creator of a profile member to help determine the search order to select among like-priority members at random.
- **Annotation String** — The annotation string enables you to identify the purpose of the profile member. The annotation can be any textual information (for example, an interface name associated with the interface identifier or a description of a service or resource associated with a group).

  Unlike the interface identifier field, the annotation string is not a search key.

Profiles are flexible; they contain members that can point to server entries, groups, and to other profiles. Profiles can also contain a special member called a *default profile member*. This optional member should point to a default profile which is usually a comprehensive backup profile that can serve the needs of most users in an organization.

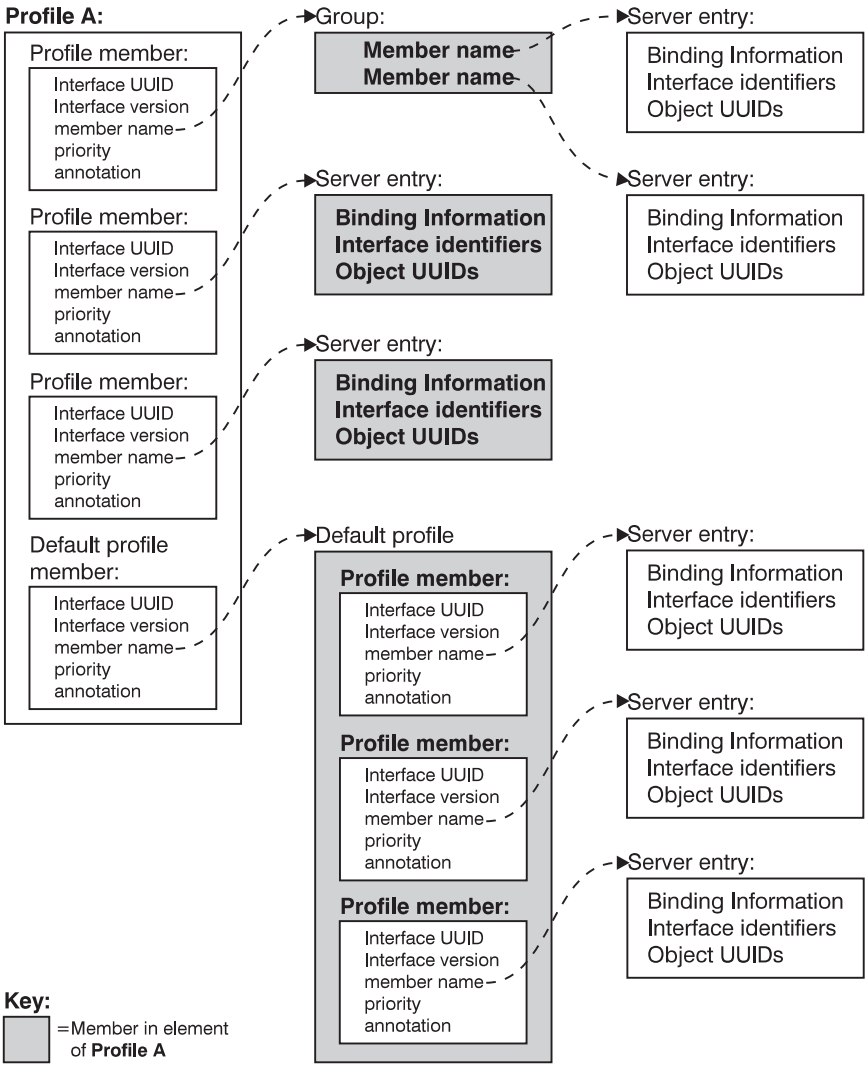


Figure 4-4. Possible Mappings of a Profile

Continuing with the printer example from the preceding discussion of group entries, the following profile entry example shows one way to use profiles to prioritize resources based on proximity to clients. In the figure, three users have personalized printer profiles that return server entries for printers nearest to them first. For example, user John is closest to laser_20 so the profile priority 1 returns that binding first. John is furthest from laser_23 so the profile priority 4 returns that binding last.

```
Profile entry name: /.:/admin/finance/accts_receivable_printers/johns_profile
  /.:/admin/finance/accts_receivable/laser_20  1
  /.:/admin/finance/accts_receivable/laser_21  2
  /.:/admin/finance/accts_receivable/laser_22  3
  /.:/admin/finance/accts_receivable/laser_23  4

Profile entry name: /.:/admin/finance/accts_receivable_printers/pats_profile
  /.:/admin/finance/accts_receivable/laser_20  3
  /.:/admin/finance/accts_receivable/laser_21  4
  /.:/admin/finance/accts_receivable/laser_22  2
  /.:/admin/finance/accts_receivable/laser_23  1

Profile entry name: /.:/admin/finance/accts_receivable_printers/wills_profile
  /.:/admin/finance/accts_receivable/laser_20  2
  /.:/admin/finance/accts_receivable/laser_21  1
  /.:/admin/finance/accts_receivable/laser_22  3
  /.:/admin/finance/accts_receivable/laser_23  4
```

To conclude this example, let us say that your department's server is being overused by another department. You could further limit its use by lowering the server's priority value in the foreign department's profile that points to your server.

Just as application servers can manage their own profile entries in CDS, they can also manage their own profile entries. However, you may find it more convenient (and more straightforward) to manually add, remove, or change server information in a profile entry. Like managing server entries and group entries, there are several methods for managing profile entries in CDS:

- Profile entry names can be hard coded into an application. You can change profile entry information in the source code but you need to recompile and rerun the application before the entry names take effect.
- Profile entry names can be passed to an application through environment variables or arguments. These methods are more convenient than recompiling but you might need to restart an application to use either method.
- Profile entry names can be directly managed in CDS by using the DCE control program's **rpcprofile** object. This manual method does not require recompiling or restarting applications.

The next sections discuss how to use the **rpcprofile** object to manually manage profile entries in CDS.

Creating a New Profile

You can create an empty profile entry in CDS by using a **rpcprofile create** operation. While profile creation is frequently performed by applications that first use a profile entry, creating an entry yourself establishes you as the owner of the entry. As the owner, you have ultimate control over who can export and manage information in the entry.

Adding a Profile Member

You can use an **rpcprofile add** operation to add a member to a profile entry. If the profile entry does not exist, the operation creates the profile entry and adds the member. The member can be a server entry or another profile entry.

Viewing the Members of a Profile Entry

You can simply list the members of a profile entry by using an **rpcprofile list** operation. This is useful for troubleshooting or for just seeing how servers are distributed in profile entries.

You can view the complete information stored with a profile entry by using an **rpcprofile show** operation. This shows the priority and the interface UUIDs associated with a member.

Importing Binding Information from a Profile Entry in CDS

Application client programs can automatically import server binding information from CDS and use it in their quest to find and communicate with a server. But occasionally, an administrator might want to import a binding. In the case where a client lacks access to CDS, it could still communicate with the server if you supplied the client with a valid binding.

Removing Members from a Profile Entry in CDS

Over time, organizational changes can require to redeploy servers in your DCE cell. You might, for instance, want to move server entries from one profile entry into another.

Use an **rpcprofile remove** operation to remove one or more members from a profile.

Deleting a Profile Entry from CDS

Organization changes or server redeployments can make some profiles obsolete. When you want to remove a profile entry from CDS, use an **rpcprofile delete** operation.

Examples — Managing Entries, Groups, and Profiles in CDS

- The following example of a binding (shown in **dcecp** syntax) indicates the server is on the host with internet address 120.101.13.157 and is available using the User Datagram Protocol (UDP).

  ```
  {nacdg_ip_udg 120.101.13.157}
  ```

- Use an **rpcentry create** operation to create an empty server entry as illustrated in the following example which creates an entry named **/.:/subsys/applications/bbs_server**. The CDS directory **/.:/subsys/applications** must already exist for this operation to succeed.

  ```
  dcecp> rpcentry create /.:/subsys/applications/bbs_server
  dcecp>
  ```

- Use an **rpcentry delete** operation to remove a server entry from CDS as shown in the following example:

  ```
  dcecp> rpcentry delete /.:/subsys/applications/bbs_server
  dcecp>
  ```

- The following example illustrates exporting a server's binding information to a server entry named **/.:/subsys/applications/bbs_server**. The object UUID identifies the data file resource used by the bbs_server.

  ```
  dcecp> rpcentry export /.:/subsys/applications/bbs_server
                  -interface {458ffcbe-98c1-11cd-bd93-0000c08adf56 1.0}
                  -binding {ncacn_ip_tcp 130.105.1.227}
                  -object {76030c42-98d5-11cd-88bc-0000c08adf56}
  dcecp>
  ```

- Use an **rpcentry import** operation to return a server's binding information.

  ```
  dcecp> rpcentry import /.:/subsys/applications/bbs_server
                  -interface {458ffcbe-98c1-11cd-bd93-0000c08adf56 1.0}
  {ncacn_ip_tcp 130.105.1.227}
  dcecp>
  ```

- You can use an **rpcentry show** operation to view the information in a server entry as illustrated in the following example. The returned information includes the interface identifier, two bindings over which the server can be reached and an object UUID of a resource maintained by the server.

```
dcecp> rpcentry show /.:/subsys/applications/bbs_server
{458ffcbe-98c1-11cd-bd93-0000c08adf56 1.0
  {ncadg_ip_udp 130.105.1.227}
  {ncacn_ip_tcp 130.105.1.227}}
{76030c42-98d5-11cd-88bc-0000c08adf56}
dcecp>
```

- Use an **rpcentry unexport** operation to remove server binding information as shown in the following example. Notice that the object UUID is not removed from the server entry unless you specify it as an option to the **unexport** operation.

```
dcecp> rpcentry unexport /.:/subsys/applications/bbs_server
                    -interface {458ffcbe-98c1-11cd-bd93-0000c08adf56 1.0}
dcecp>  rpcentry show /.:/subsys/applications/bbs_server
{76030c42-98d5-11cd-88bc-0000c08adf56}
dcecp>
```

- To create an empty group entry in CDS, use an **rpcgroup create** operation as in the following example.

```
dcecp> rpcgroup create /.:/subsys/applications/admin_bbs_servers
dcecp>
```

- To add a member to the /**.:/subsys/applications/admin_bbs_servers** group entry in CDS, use an **rpcgroup add** operation as in the following example.

```
dcecp> rpcgroup add /.:/subsys/applications/admin_bbs_servers
               -member /.:/subsys/applications/bbs_server4
dcecp>
```

- To list the members of a group entry in CDS, use an **rpcgroup list** operation as in the following example which lists the members of the group /**.:/subsys/applications/admin_bbs_servers**.

```
dcecp> rpcgroup list /.:/subsys/applications/admin_bbs_servers
/.../my_cell.goodco.com/subsys/applications/bbs_server3
/.../my_cell.goodco.com/subsys/applications/bbs_server4
dcecp>
```

- You can use an **rpcgroup import** operation to return a server's binding information. You must specify an interface using the **–interface** option as shown in the following example.

```
dcecp> rpcgroup import /.:/subsys/applications/admin_bbs_servers
                  -interface {458ffcbe-98c1-11cd-88bc-0000c08adf56 1.0}
{ncacn_ip_tcp 130.105.1.227}
dcecp>
```

- Use an **rpcgroup remove** operation to remove one or more members from a group. The following example removes **bbs_server3** from the group /**.:/subsys/applications/admin_bbs_servers**:

```
dcecp> rpcgroup remove /.:/subsys/applications/admin_bbs_servers
              -member /.../my_cell.goodco.com/subsys/applications/bbs_server3
dcecp> rpcgroup list /.:/subsys/applications/admin_bbs_servers
/.../my_cell.goodco.com/subsys/applications/bbs_server4
/.../my_cell.goodco.com/subsys/applications/bbs_server5
/.../my_cell.goodco.com/subsys/applications/bbs_server6
dcecp>
```

- The following example illustrates removing an obsolete group entry called /**.:/subsys/admin/temporaries/wp_services** from CDS.

```
dcecp> rpcgroup delete /.:/subsys/admin/temporaries/wp_services
dcecp>
```

- To create an empty profile entry in CDS use an **rpcprofile create** operation as in the following example.

```
dcecp> rpcprofile create /.:/subsys/applications/admin_group_profile
dcecp>
```

- To add a member to the /.:/**subsys**/**applications**/**wards_profile** profile entry in CDS, use an **rpcprofile add** operation as in the following example which adds the server entry /.:/**subsys**/**applications**/**bbs_server3** with a priority of 2.

```
dcecp> rpcprofile add /.:/subsys/applications/wards_profile
                -member /.:/subsys/applications/bbs_server3
                -interface {458ffcbe-98c1-11cd-88bc-0000c08adf56 1.0}
                -priority 2
dcecp>
```

- To list the members of a profile entry in CDS, use an **rpcprofile add** operation as in the following example which lists the members of the profile /.:/**subsys**/**applications**/**admin_group_profile**.

```
dcecp> rpcprofile list /.:/subsys/applications/wards_profile
/.../my_cell.goodco.com/subsys/applications/admin_bbs_servers
/.../my_cell.goodco.com/subsys/applications/bbs_server
dcecp>
```

- The following example shows all of the information contained in the profile named /.:/**cell-profile**.

```
dcecp> rpcprofile show /.:/cell-profile
{{d46113d0-a848-11cb-b863-08001e046aa5 2.0} /.../cell.co.com/sec 0 rs_bind}
{{0d7c1e50-113a-11ca-b71f-08001e01dc6c 1.0} /.../cell.co.com/sec-v1 0 secidmap}
{{8f73de50-768c-11ca-bffc-08001e039431 1.0} /.../cell.co.com/sec 0 krb5rpc}
{{b1e338f8-9533-11c9-a34a-08001e019c1e 1.0} /.../cell.co.com/sec 0 rpriv}
{{b1e338f8-9533-11c9-a34a-08001e019c1e 1.1} /.../cell.co.com/sec 0 rpriv}
{{6f264242-b9f8-11c9-ad31-08002b0dc035 1.0} /.../cell.co.com/lan-profile 0 LAN}
{{4d37f2dd-ed43-0000-02c0-37cf2e000001 4.0} /.../cell.co.com/fs 0 fs}
{{eb814e2a-0099-11ca-8678-02608c2ea96e 4.0} /.../cell.co.com/subsys/dce/dfs/bak 0 bak}
dcecp>
```

- You can use an **rpcprofile import** operation to return a server's binding information. You must specify an interface using the **–interface** option as shown in the following example.

```
dcecp> rpcprofile import /.:/subsys/applications/wards_profile
                  -interface {458ffcbe-98c1-11cd-88bc-0000c08adf56 1.0}
{ncacn_ip_tcp 130.105.1.202}
{ncacn_ip_tcp 130.105.1.227}
dcecp>
```

  You can use other options such as **–version** and **–object** to further specify a binding. Use the **–max** option to limit the number of bindings returned as shown in the following example.

```
dcecp> rpcprofile import /.:/subsys/applications/wards_profile
                  -interface {458ffcbe-98c1-11cd-88bc-0000c08adf56 1.0}
                  -max 1
{ncacn_ip_tcp 130.105.1.202}
dcecp>
```

- The following example removes member /.:/**subsys**/**applications**/**admin_bbs_servers** from the profile /.:/**subsys**/**applications**/**wards_profile**:

```
dcecp> rpcprofile remove /.:/subsys/applications/wards_profile
                  -member /.:/subsys/applications/admin_bbs_servers
                  -interface {458ffcbe-98c1-11cd-88bc-0000c08adf56 1.0}
dcecp>
```

- The following example illustrates removing an obsolete profile entry called /.:/**subsys**/**admin**/**temporaries**/**74232_profile** from CDS.

```
dcecp> rpcprofile delete /.:/subsys/admin/temporaries/74232_profile
dcecp>
```

Monitoring Your Program's Performance

Introduction — Monitoring Your Program's Performance

You can instrument your DCE application to determine many aspects of its performance. Under DCE, you can monitor clients and servers separately or you can monitor several instances of an application (client or server) at the same time. Using the monitoring subsystem, you can measure statistics such as TCP and UDP call counts, marshalling and unmarshalling time, and elapsed time. You can also measure many other process- and system-level statistics.

More Detailed Information — Monitoring Your Program's Performance

To initialize and start the DCE instrumentation subsystem, your DCE application must contain the following line for the client code:

`IF_client_inst_init(`*`program name`*`)`

or, for the server code, the application must contain the following line:

`IF_server_inst_init(`*`program name`*`)`

where *program name* is usually the name of the program (for example, **(argv[0])**). However, you can use any convention.

When recompiling your DCE application, include the flags **–spmi csempq** on the IDL compiler line in your makefile. The IDL compiler generates instrumented stub files which, in turn, must be linked with DCE instrumentation library routines. The DCE instrumentation fileset must be installed to make these routines available.

For more information on how to instrument your DCE applications for monitoring, please see the README file in **/usr/lpp/dce/examples/inst**.

Routing Serviceability Messages

Introduction — Routing Serviceability Messages

The DCE serviceability mechanism is designed to be used mainly for server informational and error messaging — that is, for messages that are of interest to those who are concerned with server maintenance and administration (in the broadest sense of these terms). The essential idea of the mechanism is that all server events that are significant for maintaining or restoring normal operation should be reported in messages that are made to be self-documenting. As a result, (assuming that all events have been correctly identified and reported) users and administrators will always be able to learn what action they should take in a given situation.

Note: User-prompted, interactive, client-generated messaging is handled through the standard DCE messaging interface.

More Detailed Information — Routing Serviceability Messages

The serviceability component is used by the DCE components (RPC, DTS, Security, and so on) for their own server messaging, and it is made available as an API for use by DCE application programmers who want to standardize their applications' server messaging. The serviceability API is described in the *DCE for AIX Application Development Guide — Core Components*.

Messaging uses XPG4 message catalogs to hold message texts, but it adds an additional layer to the XPG4 functionality. The message catalogs and other required data (and documentation) files are generated by a utility called *sams* (Symbols And Message Strings). Its input is a text file that establishes

some organizational information about the program that is to use the messages, followed by a series of specifications of the messages themselves. The serviceability mechanism allows system administrators to control the routing of these messages. Specifically, you can define message routings based on the severity levels (FATAL, ERROR, and so on) defined for the messages.

The following sections describe how to control the routing of serviceability messages. First, it provides an overview of serviceability messaging in the DCE. It then describes how you can use message severity levels to control routing. Finally, it describes the different ways you can specify routing for serviceability messages.

Serviceability Message Severity Levels

Serviceability messages are categorized by their severity level, which provides important information about the situation that causes the program to issue the message. Every message's severity is defined in the text of the message itself (for example, `NOTICE` indicates that a message is an informational notice), and system administrators can route messages differently on the basis of their severity levels.

The following table lists the possible severity levels and provides an explanation for each.

Table 4-1. Serviceability Message Severity Levels

Name	Meaning
FATAL	Fatal error exit: An unrecoverable error (such as database corruption) has occurred and will probably require manual intervention to be corrected. The program usually terminates immediately after such an error.
ERROR	Error detected: An unexpected event that is nonterminal (such as a timeout), or is correctable by human intervention, has occurred. The program will continue operation, although some functions or services may no longer be available. This severity level may also be used to indicate that a particular request or action could not be completed.
WARNING	Correctable error: An error occurred that was automatically corrected (for example, a configuration file was not found and default values were used instead). This severity level may also be used to indicate a condition that *may* be an error if the effects are undesirable (for example, removing all files as a side-effect of removing a nonempty directory). This severity level may also be used to indicate a condition that, if not corrected, will eventually result in an error (for example, a printer's running low on paper).
NOTICE	Information notice: A significant routine major event has occurred (for example, a server has started).
NOTICE_VERBOSE	Verbose information notice: A significant routine event has occurred (for example, a directory entry was removed).

How to Route Serviceability Messages

Serviceability messages can be written to any of the normal output destinations. You can specify routing for serviceability messages in any of the following four ways:

- Through the **dcecp log** object, if the server supports the remote serviceability interface
- By the contents of a routing file
- By the contents of an environment variable
- By command line flags (usually **–w**), if supported by the server

Note: Each of the methods accepts the string syntax form for serviceability routing specifications. In addition, **dcecp** allows you to use Tcl (Tool Command Language) syntax, which is easier to use when writing scripts.

Routing a message actually consists of specifying two things:

- How the message should be processed (that is, the form it should be put in), and
- Where the message should be sent (its destination).

The two specifications are sometimes closely interrelated, and sometimes specifying a certain destination implies that the message must be put into a certain form. This fact allows certain combinations to be abbreviated.

The ways to route serviceability messages are described separately in the following sections.

Using the dcecp log Object

The **dcecp log** object represents the current state of routing for DCE serviceability messages for a given server. The **log** object supports both serviceability routing and debug routing.

The **log** object exports a number of operations. The following operations are useful for serviceability message routing:

- **log list** returns a list of the components registered by the server. The **–comp** option allows you to also return a list of the subcomponents for one or more named components.
- **log show** returns a list describing the current serviceability routing specifications for a server.
- **log modify** sets message routing specifications for one or more specified servers.

For a complete description of the **dcecp log** object and the syntax for its supported operations, refer to the **log** reference page in the *DCE for AIX Administration Command Reference*.

The remainder of this section describes only the **log modify** operation and how to use it to establish routings for serviceability messages. Remember that routing is always set on a per-server basis and is recorded in the **log** object for each server.

The syntax for the **log modify** operation is:

```
log modify {string_binding_to_server | RPC_server_namespace_entry }
          {-change serviceability_routing_specifications}
```

You can specify multiple target servers as a space-separated list. Specify each server by supplying either the RPC string binding that describes the server's network location (*string_binding_to_server*) or a namespace entry of the server (*RPC_server_namespace_entry*). When specifying multiple servers, you can mix the forms in the same list.

A *serviceability_routing_specification* is a space-separated list of serviceability routing elements. No spaces are allowed within the specification of an individual routing element. Each routing element is a sub-string consisting of four fields containing PCS (Portable Character Set) data, as follows (shown in string syntax form):

severity:*output_form*:*destination*[:*application-defined*]

where:

severity Is one of the message severity levels — **FATAL**, **ERROR**, **WARNING**, **NOTICE**, or **NOTICE_VERBOSE**.

output_form Specifies how messages of the associated severity level should be processed, and must be one of the following:

BINFILE Write these messages as binary log entries

TEXTFILE Write these messages as human-readable text

FILE Equivalent to **TEXTFILE**

DISCARD Do not record these messages

STDOUT Write these messages as human-readable text to standard output

STDERR Write these messages as human-readable text to standard error

Files written as **BINFILE**s can be read and manipulated with a set of logfile APIs, which are described in the *DCE for AIX Application Development Guide — Core Components*.

The **BINFILE**, **TEXTFILE**, and **FILE** *output_form* specifiers may be followed by a two-number specifier of the form:

.*gens***.***count*

where:

gens Is an integer that specifies the number of files (i.e., generations) that should be kept

count Is an integer specifying how many entries (i.e., messages) should be written to each file

The multiple files are named by appending a dot to the simple specified name, followed by the current generation number. When the number of entries in a file reaches the maximum specified by *count*, the file is closed, the generation number is incremented, and the next file is opened. When the maximum number of files have been created and filled, the generation number is reset to 1, and a new file with that number is created and written to (thus overwriting the already-existing file with the same name), and so on. Thus the files wrap around to their beginning, and the total number of log files never exceeds *gens*, although messages continue to be written as long as the program continues writing them.

destination Specifies where the message should be sent, and is a path name. You can leave this field blank if the *output_form* specified is **DISCARD**, **STDOUT**, or **STDERR**. The field can also contain a **%ld** string in the file name which, when the file is written, will be replaced by the process ID of the program that wrote the message(s). File names may *not* contain colons (:), semi-colons (;), percent signs (%), or spaces.

application-defined
Is used for application-specific information. Standard DCE programs ignore it.

String Syntax

The string syntax for a serviceability routing specification is:

severity:*output_form*:*destination*[:*application-defined*][;...]

Tcl Syntax

The Tcl syntax for a serviceability routing specification takes the form:

{*severity output_form destination application-defined***}**

where *severity*, *output_form*, *destination*, and *application-defined* are specified as described above. In Tcl syntax, multiple routing specifications take the form:

{ {*specification***} {***specification***} {***specification***} }**

If a file called **/opt/dcelocal/var/svc/routing** exists, the contents of the file (if in the proper format) will be used to determine the routing of messages written by the serviceability mechanism.

The default location of the serviceability routing file is normally:

/opt/dcelocal/var/svc/routing

However, you can specify a different location for the file by setting the value of the environment variable **DCE_SVC_ROUTING_FILE** to the complete desired path name.

The routing file contains lines that specify the routing desired for the various kinds of messages (based on message severity level). Each line consists of three fields as follows:

severity:*output_form*:*destination*[:*application-defined*][;...]

You can supply multiple routings by specifying additional *output_form*:*destination* pairs as a semi-colon-separated list.

In the routing file, blank lines beginning with the # character are treated as comments.

Using Environment Variables

Serviceability message routing can also be specified by the contents of certain environment variables. If you use environment variables, the routings you specify will override any conflicting routings specified by a routing file.

The routings are specified (on the basis of severity level) by putting the desired routing instructions in the following environment variables:

- **SVC_FATAL**
- **SVC_ERROR**
- **SVC_WARNING**
- **SVC_NOTICE**
- **SVC_NOTICE_VERBOSE**
- **SVC_BRIEF**

Each variable should contain a single string in the format:

severity:*output_form*:*destination*[:*application-defined*][;...]

You can supply multiple routings by specifying additional *output_form*:*destination* pairs as a semi-colon-separated list.

Examples — Routing Serviceability Messages

- You can define multiple routing specifications as a semi-colon separated list. For example, this specification:

  ```
  FATAL:TEXTFILE:/dev/console;STDOUT:
  ERROR:TEXTFILE.5.100:/tmp/errors
  EXIT:DISCARD:
  *:FILE:/tmp/svc-log
  NOTICE:BINFILE:/tmp/log%ld
  WARNING:STDOUT:
  ```

  instructs the serviceability mechanism to:

  – Send fatal error messages to the console and to standard output.

  – Send other error messages to a log-rolled file.

  – Discard normal exit reports.

  - Write all messages to a log file.
  - Send informational messages to a temporary binary log.
  - Send warnings to standard output.

- The specification examples shown in string format above would be expressed in Tcl syntax as follows:

```
{FATAL { {TEXTFILE /dev/console} STDOUT} }
{ERROR TEXTFILE.5.100 /tmp/errors}
{EXIT DISCARD}
{* FILE /tmp/svc-log}
{NOTICE BINFILE /tmp/log%ld }
{WARNING STDOUT {} }
```

Chapter 5. Cell Directory Service (CDS)

This section describes the control program tasks, utilities, and processes. For introductory information on the DCE control program, see “Cell Directory Service (CDS) Overview” on page 1-28. That section also groups related tasks together in the order in which they may normally occur so you can access these referenced tasks in a logical, tutorial-type manner. You may find this helpful in performing related tasks together from beginning to end.

ACL Entry Types Used for Principals

Use ACL entry types to specify the category of principal for which the ACL entry is created. These ACL entry types are described below:

Entry Type	Purpose
user	Specifies an ACL entry for an individual principal whose credentials were authenticated within the local cell.
group	Specifies an ACL entry for an authorization group whose members are defined within the local cell.
other_obj	Specifies an ACL entry for authenticated principals in the local cell who are not individual users named by an ACL entry of the type **user** or members of a group named by an ACL entry of the type **user**.
foreign_user	Specifies an ACL entry for authenticated principals in a foreign cell who are not individual users named by an ACL entry of the type **foreign_user** or members of a group named by an ACL entry of the type **foreign_group**.
foreign_group	Specifies an ACL entry for an authorization group whose members are defined in another named cell.
foreign_other	Specifies an ACL entry for all principals whose credentials were authenticated in another named cell, unless they are specifically named in ACL entries of type **user** or are members of an authorization group that is named in ACL entries of type **group**.
mask_obj	Specifies an ACL entry containing a mask that is substituted for the permissions of any principals, whose credentials are either authenticated or unauthenticated.
unauthenticated	Specifies an ACL entry for principals who cannot pass authentication procedures.
any_other	Specifies an ACL entry for an authenticated principal who is not otherwise covered by any of the preceding ACL entry types.
user_delegate	Specifies an ACL entry for an intermediary that acts for an authenticated principal in the local cell.
group_delegate	Specifies an ACL entry for an intermediary that acts for the authenticated principals who are members of an authorization group in the local cell.
other_delegate	Specifies an ACL entry for an intermediary that acts for authenticated principals in the local cell who are not individual users named by an ACL entry of the type **user_delegate** or who are not members of a group named by an ACL entry of the type **group_delegate**.
foreign_user_delegate	Specifies an ACL entry for an intermediary that acts for an authenticated principal in a foreign cell.

foreign_group_delegate
Specifies an ACL entry for an intermediary that acts for the members of an authorization group in a foreign cell.

foreign_other_delegate
Specifies an ACL entry for an intermediary that acts for authenticated principals in a foreign cell who are not individual users named by an ACL entry of the type **foreign_user_delegate** or members of a group named by an ACL entry of the type **foreign_group_delegate**.

any_other_delegate
Specifies an ACL entry for an intermediary that acts for authenticated principals in the local cell or in a foreign cell who are not named by an ACL entry of any other type for intermediaries of authenticated principals or groups.

Note: The delegate ACL types are only supported if the CDS directories are upgraded to Version 4.0 (the default is version 3.0, which is compatible with older DCE product versions). See "Upgrading the Directory Version on the Cell Root Directory" under "More Detailed Information — Creating Directories" on page 5-16 for information on performing the upgrade.

ACL Types Supported by CDS

Introduction — ACL Types Supported by CDS

CDS supports the following DCE ACL types:

- **Object ACL** — You can use the Object ACL type to grant permissions to any CDS name (that is, object entries, soft links, child pointers, clearinghouses, and directories), as well as to CDS clerks and servers. When associated with a CDS directory, the permissions you grant with the Object ACL type apply only to the directory itself, not to the directory's contents or to any child directories.
- **Initial Object Creation ACL** — This ACL type applies only to CDS directory names. Use this ACL type to grant permissions specifically to a directory's future contents, including soft links, application-defined object entries, child pointers, and clearinghouse object entries. The permissions you grant by using the Initial Object Creation ACL type apply only to the future contents of the directory, not to the directory itself. The permissions are inherited only by names that are created in the directory *after* you create the ACL entry. Permissions are not propagated to names that already exist in the directory.
- **Initial Container Creation ACL** — This ACL type applies only to CDS directory names. Use this ACL type to grant permissions to a directory that automatically propagate (the default) to all child directories that you can later create under that directory. The permissions you grant by using the Initial Container Creation ACL type are inherited only by the child directories that you create *after* you create the ACL entry; permissions are not propagated to child directories that already exist.

Steps — ACL Types Supported by CDS

- To edit an Initial Object Creation ACL, specify the **–io** option of the **dcecp** program's **acl modify** command.
- To edit an Initial Container Creation ACL, specify the **–ic** option of the **dcecp** program's **acl modify** command.

An In-Depth Analysis of DCE Names

Introduction — An In-Depth Analysis of DCE Names

The remainder of this section provides a detailed description of the different kinds of names that make up the DCE namespace.“Valid Characters and Naming Rules for CDS” on page A-48 and “Additional Rules” on page A-1 contain further details about valid characters and naming conventions in CDS, GDS, and DNS names.

More Detailed Information — An In-Depth Analysis of DCE Names

CDS Names

Every cell contains at least one system that is running a CDS server. A CDS server stores and maintains names and handles requests to create, modify, and look up data. The total collection of names shared by CDS servers in a cell is called a *cell namespace*. The cell namespace administrator can organize CDS names into a hierarchical structure of directories. CDS directories, which are conceptually similar to the directories in your operating system's file system, are a logical way to group names for ease of management and use.

In a cell namespace, any directory that has a directory beneath it is considered the parent of the directory beneath it. Any directory that has a directory above it is considered a child of the directory above it. The top level of the cell namespace is called the *cell root*. You can refer to the cell root either by the global name of the cell or by the short-form /**.:** prefix.

Figure 5-1 shows a simple cell namespace hierarchy, starting at the cell root. The cell root (/**.:**) is the parent of the directories named /**.:/hosts** and /**.:/subsys**. The /**.:/subsys** directory is a child of the cell root directory and the parent of the /**.:/subsys/dce** directory.

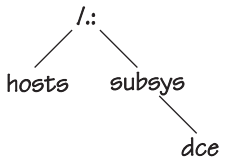


Figure 5-1. Sample CDS Namespace Hierarchy

The complete specification of a CDS name, going left to right from the cell root to the entry being named, is called the *full name*. Each element within a full name is separated by a slash (/) and is called a *simple name*. For example, suppose the /**.:/hosts** directory shown in Figure 5-1 contains an entry for a DCE host named **bargle**. The CDS full name for that entry is /**.:/hosts/bargle**. Multiple consecutive slashes (//) are turned into a single slash (/) in a full name.

Multiple directory levels enable flexibility in distributing, controlling access to, and managing many names. A directory hierarchy also reduces the probability of duplicate names. For example, /**.:/subsys/Hypermax/printQ/server1** and /**.:/subsys/ABC/spell/server1** are unique names.

DNS Names

The DCE naming environment supports the version of DNS that is based on Internet Request For Comments (RFC) 1034 and RFC 1035. Many networks currently use DNS primarily as a name service for host names. The most commonly used implementation of DNS is the Berkeley Internet Naming Domain (BIND). The BIND namespace is a hierarchical tree with its topmost levels under the control of the Network Information Center (NIC). See “Global and Cell Considerations” on page 1-4 for information on how to contact the NIC Domain Registrar to register a domain name.

The names directly under the root of the BIND namespace include two-letter codes for countries (such as **us** and **gb**) as defined in ISO Standard 3166, "Codes for the Representation of Names of Countries." Other names one level below the root include several generic administrative categories, such as **com** (commercial), **edu** (educational), **gov** (government), and **org** (other organizations). The owners of these names can grant permission to companies and organizations to create new subordinate names.
Figure 5-2 shows a sample portion of the BIND namespace. (The double quotes indicate that the root of the namespace has a null name and is not addressable.) Note that, like CDS names, DNS names are not typed (that is, they do not consist of pairs of attribute types and values).

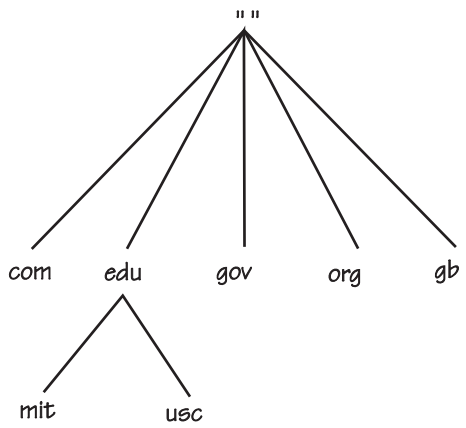


Figure 5-2. Sample Portion of the BIND Namespace

A DNS name consists of a string of hierarchical names that are separated by dots (.) and arranged right to left from the root of the namespace. For example, the name **ai.mit.edu** represents the branch of the namespace owned by the Massachusetts Institute of Technology artificial intelligence department. Note that the order of elements in the name is the reverse of the order for CDS and GDS names.

To use a DNS cell name as part of a global DCE name, specify the DNS name intact between two slashes. For example, a cell whose DNS name is **ai.mit.edu** might contain a directory whose CDS name is /**.:/profiles**. Users should enter /**.../ai.mit.edu/profiles** to refer to the directory by its global name.

Names Outside of the DCE Directory Service

Not all DCE names are stored directly in the DCE Directory Service. Some services connect into the cell namespace by means of specialized CDS entries called *junctions*. A junction entry contains binding information that enables a client to connect to a server outside of the Directory Service.

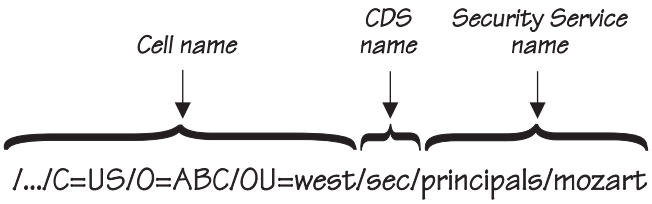
The DCE Security Service keeps a database of principals (users and servers) and information about them, such as their passwords. The default name of the Security Service junction is /**.:/sec**.

DFS is another service that uses junctions. The DFS Fileset Location Service keeps a database that maps DFS filesets to the servers where they reside. The junction to this database has a default name of /**.:/fs**.

The DCE namespace is a connected tree of many kinds of names from many different sources. The GDA component of the Directory Service provides connections out of the cell and to other cells through a global namespace, such as GDS or DNS. In a similar manner, junctions enable connections downward from the cell namespace to other services.

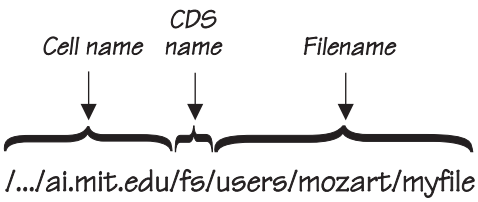
Examples — An In-Depth Analysis of DCE Names

- The following example illustrates the parts of a global DCE principal name:



  The cell name, /**.../C=US/O=ABC/OU=west**, is a GDS name. The **sec** portion is the junction entry in CDS, and **principals/mozart** is a principal name that is stored in the Security Service database.

- The following example illustrates the parts of a global DCE file name:



  The global name contains a DNS cell name, /**.../ai.mit.edu**. The **fs** portion is the file system junction entry in CDS and /**users/mozart/myfile** is the name of a file.

Backing Up Namespace Information

Introduction — Backing Up Namespace Information

Because updates and skulks of directories can occur asynchronously, and because of the distributed nature of a namespace, you cannot always depend on traditional backup methods to preserve CDS data. The remainder of this section describes when to use directory replication and operating system backups.

More Detailed Information — Backing Up Namespace Information

Using Replication to Back Up Namespace Information

Directory replication is always the most reliable way to back up the information in your namespace. When you create a new replica of a directory at a clearinghouse, you are not only distributing the information, but also creating an up-to-date, real-time backup of the information. If a replica in one clearinghouse becomes unavailable, users can look up the information they need in another replica of the directory in some other clearinghouse. The more replicas of a directory you create, the more likely users will always be able to find the information contained in the directory somewhere in the namespace.

If an entire clearinghouse is corrupted, you can restore it by creating a new clearinghouse and then creating new replicas of the directories that were stored there. See “Creating a Read-Only Replica” on page 5-17 for complete information on how to create a replica.

Using Operating System Backups

Because a namespace is a distributed database to which modifications are synchronized at variable intervals, any traditional backup of a particular server system always contains old and incomplete information. If you frequently create, modify, or delete names, restoring an out-of-date backup can cause

recently created names to disappear, recent modifications to be reversed, or recently deleted names to reappear in the namespace. The degree to which a traditional backup reflects the current condition of a clearinghouse depends entirely on:

- How recently the backup was created,
- What modifications were made since that time, and
- Whether the backup included the clearinghouse files in the directory **/opt/dcelocal/var/directory/cds**.

If you decide to use operating system backups, you need to back up only the server systems whose clearinghouses store master replicas of directories. To ensure that you back up your namespace completely, check that:

- The servers on these systems are disabled by using the **cdscp** program's **disable server** command, and
- The files in the root directory **/opt/dcelocal/var/directory/cds** are included in the backup.

If your namespace is small enough to be maintained in one clearinghouse, you can reliably use traditional operating system backups to save and restore the clearinghouse data. If only one clearinghouse exists, only one replica (the master replica) of each directory exists. This eliminates the need to account for the discrepancies that may exist among multiple directory replicas. Remember that the more frequently you back up clearinghouse data, the more up-to-date that information will be if you need to restore it.

CDS Control Program Commands and Required Permissions

The following lists all the **dcecp** commands that operate on CDS objects with the permissions that a principal must have to execute the commands.

Commands	Required
directory add	Insert permission to the parent directory where the child pointer (**–member** option) is to be placed.
directory create	For a new directory: Insert and read permissions to the parent directory and write permission to the clearinghouse that stores the master replica of the new directory. Also, the server principal needs read and insert permissions to the parent directory of the new directory. For a replica of an existing directory (**–replica** option): Administer permission to the directory that you intend to replicate and write permission to the clearinghouse (**–clearinghouse** option) that stores the new replica. Also, the server principal needs read, write, and administer permissions to the directory that you intend to replicate.
directory delete	For a directory: Delete permission to the directory and write permission to the clearinghouse that stores the master replica of the directory. Also, the server principal needs administer permission to the parent directory or delete permission to the child pointer that points to the directory you intend to delete. For a replica of an existing directory: Administer permission to the directory whose replica (**–replica** option) you want to delete and write permission to the clearinghouse (**–clearinghouse** option) from which you are deleting the replica.
directory list	Read permission to the directory whose contents you want to list.
directory modify	Write permission to the directory to which you want to add (**–add** option) or change (**–change** option) the attribute or attribute value, and delete permission to the directory from which you want to remove the attribute or attribute value (**–remove** option).

directory synchronize
Administer, write, insert, and delete permission to the directory. Also, the server principal needs administer, read, and write permissions to the directory.

directory show Read permission to the directory whose attributes you want to list. If you specify a global file-name character, you also need permission to the directory's parent directory.

For a replica of a directory (**–replica** option): Read permission to the directory of which the replica is a member.

For a child directory (**–member**): Read permission to the child.

object create Insert permission to the parent directory which is to store the object entry.

object delete Delete permission to the object entry or administer permission to the parent directory that stores the object entry.

object modify Write permission to the object entry for which you want to add (**–add** option) or change (**–change** option) the attribute or attribute value, and delete permission to the object entry or administer permission to the parent directory of the object for which you want to remove (**–remove** option) the attribute or attribute value.

object show Read permission to the object entry whose attributes you want to list.

cdscache create Write permission to the clerk that is to create the server entry in the local CDS cache.

cdscache delete Write permission to the clerk that is to delete the server entry in the local CDS cache.

cdscache dump Superuser (**root**) privileges on the clerk system where the CDS cache resides. No CDS permissions are required.

cdscache show Read permission to the clerk that is to retrieve the server (**–server** option) or clearinghouse (**–clearinghouse** option) information from the CDS cache.

clearinghouse create
Write permission to the server on which you intend to create the clearinghouse and administer permission to the cell root directory. Also, the server principal needs read, write, and administer permissions to the cell root directory.

clearinghouse delete
Write and delete permissions to the clearinghouse to be deleted and administer permission to all directories that store replicas in the clearinghouse. Also, the server principal needs delete permission to the associated clearinghouse object entry and administer permission to all directories that store replicas in the clearinghouse.

clearinghouse disable
Write permission to the server that is to loose knowledge of the clearinghouse.

clearinghouse show
Read permission to the clearinghouse whose attributes you want to list. If you specify a global file-name character in the clearinghouse name, you also need read permission to the cell root directory.

link create Insert permission to the directory in which you intend to create the soft link.

link delete Delete permission to the soft link or administer permission to the directory that stores the soft link to be deleted.

link modify Write permission to the soft link whose attributes are to be modified.

link show Read permission to the soft link whose attributes are to be listed.

The following lists commands for operating on CDS objects that are only provided by the CDS control program (**cdscp**), along with their execute permissions.

Commands	Required Permissions
disable clerk	Write permission to the clerk.
disable server	Write permission to the server.
set cdscp confidence	No permissions are needed.
set directory to new epoch	Administer permission to the directory. The server principal needs administer, read, and write permissions to the directory. When designating a new master replica, you also need write permission to the clearinghouse that stores the new master replica and the server principal needs write permission to each clearinghouse where the replica type is changed to read only.
show cdscp confidence	No permissions are needed.
show cell	Read permission to the cell root directory.
show clerk	Read permission to the clerk.
show server	Read permission to the server.

CDS Operations

Operation of the Cell Directory Service involves the following major participants:

- Client Applications
- Servers
- Clerks
- Clearinghouses

CDS uses a client/server model. An application that depends on CDS to store and retrieve information for it is a client of CDS. Client applications create names for resources on behalf of their users. Through a client application, a user can supply other information for CDS to store as attributes of a name. Then, when a client application user refers to the resource by its CDS name, CDS retrieves data from the attributes for use by the client application.

A system running CDS server software is a CDS server. A CDS server stores and maintains CDS names and handles requests to create, modify, or look up data.

A component called the *clerk* is the interface between client applications and CDS servers. Every DCE node must run a CDS clerk. The clerk receives a request from a client application, sends the request to a server, and returns the resulting information to the client. This process is called a *lookup*. The clerk is also the interface through which client applications create and modify names. One clerk can work on behalf of many client applications.

The clerk caches (saves) the results of lookups so that it does not have to repeatedly go to a server for the same information. The cache is written to disk periodically so that the information can survive a system reboot or the restart of an application. When you stop the CDS advertiser, the cache is written to disk. Caching improves performance and reduces network traffic.

Figure 5-3 on page 5-9 shows a sample configuration of CDS clerks and servers on a nine-node Local Area Network (LAN). Every node is a clerk and CDS servers run on two selected nodes.

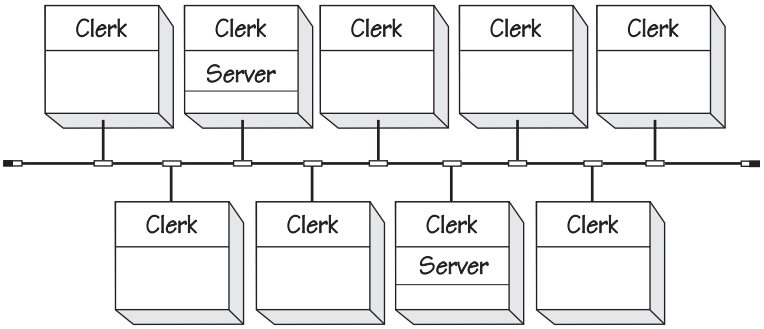


Figure 5-3. CDS Clerks and Servers on a LAN

Every CDS server has a database called a *clearinghouse* in which it stores names and other CDS data. The clearinghouse is where a CDS server adds, modifies, deletes, and retrieves data on behalf of client applications. Although more than one clearinghouse can exist at a server or node, it is not recommended as a normal configuration.

Figure 5-4 shows the interaction between a CDS client, clerk, server, and clearinghouse during a simple lookup. The figure shows the following CDS lookup steps:

1. The client application on Node 1 sends a lookup request to the local clerk.
2. The clerk checks its cache and, not finding the name there, contacts the server on Node 2.
3. The server checks to see if the name is in its clearinghouse.
4. The name exists in the clearinghouse, so the server gets the requested information.
5. The server returns the information to the clerk on Node 1.
6. The clerk passes the requested data to the client application. The clerk also caches the information so that it does not have to contact a server the next time a client requests a lookup of that same name.

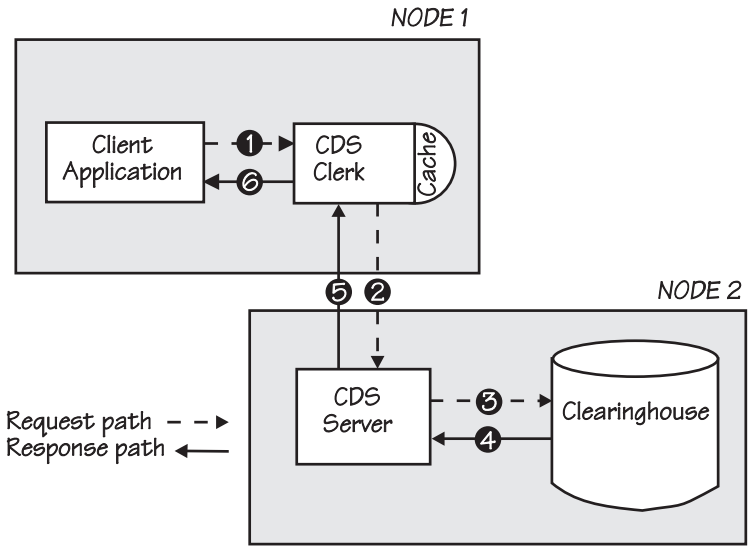


Figure 5-4. A Sample CDS Lookup

CDS Servers Access to the Namespace

Introduction — CDS Servers Access to the Namespace

To automate the process of granting all CDS servers the permissions that they require, the CDS cell configuration process creates an authorization group for CDS servers under the fixed name **subsys/dce/cds-servers**. The principal name of the initial server in the cell is added to this group as part of the configuration process. Immediately after the group is created, the configuration process (through **acl_edit**) grants full permissions (**r**, **w**, **i**, **d**, **t**, **c**, **a**) to the cell root directory of the new namespace on behalf of the group. ACL entries of the Object ACL and Initial Container Creation ACL types are created by specifying **subsys/dce/cds-servers** as the principal in each ACL entry. This ensures that the group has full access to all future directories and their contents.

Thereafter, whenever a new server is configured in the cell, the server configuration process automatically adds the principal name of the new server to the group. Through this process, all CDS servers in the cell receive adequate permissions to all directories in the namespace.

Permissions Required — CDS Servers Access to the Namespace

CDS servers require permission to the cell root directory and to lower-level directories to successfully execute the following CDS commands:

- **clearinghouse create**
- **directory create** (for directories and replicas)
- **directory delete** (for directories and replicas)
- **directory synchronize**

cdscp Program Usage

Introduction — cdscp Program Usage

At the current time, you must use the **cdscp** program, rather than the **dcecp** program, for certain CDS maintenance tasks. For example, only the **cdscp** program allows you to stop a CDS clerk (**disable clerk**) or to reconstruct a directory's replica set by changing the version number (**set directory to new epoch**).

In addition to describing the **cdscp** commands that the **dcecp** program does not currently implement, this section provides basic instructions for using the **cdscp** program.

More Detailed Information — cdscp Program Usage

The following lists the **cdscp** program commands that you use to manage CDS clerks, servers, directory replicas, and cells.

Commands	Definitions
disable clerk	Stops the execution of a CDS clerk.
disable server	Stops the execution of a CDS server.
set cdscp confidence	Permits you to set the confidence level of CDS clerk calls.
set directory to new epoch	Reconstructs a directory's replica set by designating a new master replica.

show cdscp confidence
Permits you to see the current confidence level of CDS clerk calls.

show cell Displays the information needed for creating a cell entry in DNS or GDS.

show clerk Displays the attributes of a CDS clerk.

show server Displays the attributes of a CDS server.

For detailed descriptions of all of the **cdscp** program commands, see the **cdscp** reference page.

Steps — cdscp Program Usage

- To start the **cdscp** program, enter the following command:

  ```
  $ cdscp
  ```

- To exit from the **cdscp** program, enter the following command:

  ```
  cdscp> quit
  ```

- To read a file of the **cdscp** commands while inside the control program, enter the following command:

  ```
  cdscp> do filename
  ```

Cell Directory Service User Interface

Introduction — CDS User Interface

CDS has several entities that can be managed through user interfaces provided in DCE. A CDS entity is any individually manageable piece of the Cell Directory Service. CDS directories, soft links, and object entries are the most common entities that you manage with the DCE user interfaces. Some object entries, though, are normally managed through the client application that creates them.

The **dcecp** program provides many commands for managing CDS entities. “Managing the DCE Cell Directory Services” on page 1-31 contains information about these commands.

More Detailed Information — CDS User Interface

CDS also comes with two other user interfaces — the **cdscp** program and the Browser.

- The **cdscp** program, which is available on all clerks and servers, is an interface that accepts commands targeted for these entities and directory replicas. However, most of the **cdscp** program commands have equivalents in the **dcecp** program and you are encouraged to use the **dcecp** commands instead. The small number of **cdscp** program commands that you must use for operations on CDS servers, clerks, and replicas are described in “Managing the DCE Cell Directory Services” on page 1-31.
- The Browser is a tool for viewing the content and structure of a namespace. It runs on workstations with windowing software that is based on the OSF/Motif** graphical user interface. Using a mouse to manipulate pull-down menus, you can view the directory structure of a namespace, view child directories of a particular directory, view the object entries and soft links in a directory, and set a filter to display only object entries of a particular class. (For users who do not have windowing software, similar functions are available with both the **dcecp** and **cdscp** programs.)

In addition to the **dcecp** and **cdscp** programs, other DCE user interfaces allow access to, and management of, CDS names. For example, users can control access to CDS directories and their contents by using the ACL editor (such as the **dcecp acl** object or the **acl_edit** program), which is supplied with the DCE Security Service. RPC application programmers can create server entries, groups,

and configuration profiles in the cell namespace with the **dcecp** or **rpccp** programs. For details on how to use these interfaces, see *DCE for AIX Administration Command Reference*.

Cells Determine Naming Environments

Introduction — Determining Naming Environments

In addition to delineating security and administrative boundaries for users and resources, cells determine the boundaries for sets of names. Because different naming components operate in a cell and outside of a cell, naming conventions in the cell and global environments differ as well. The DCE naming environment supports two kinds of names — global and cell-relative (local) names. The following introduces the concept of global and local names. “An In-Depth Analysis of DCE Names” on page 5-2 describes CDS, GDS, and DNS names in detail.

More Detailed Information — Determining Naming Environments

Global Names

All entries in the DCE Directory Service have a global name that is universally meaningful and usable from anywhere in the DCE naming environment. The prefix /**...** indicates that a name is global. A global name can refer to an object within a cell (named in CDS) or an object outside of a cell (named in GDS).

If a DCE administrator is establishing a hierarchy of cells during initial cell configuration, he or she must obtain a unique GDS or DNS cell name for the cell at the top of the hierarchy from the GDS or DNS global directory service authorities. All of the cells beneath this cell share this name.“Global and Cell Considerations” on page 1-4 provides details on how to obtain GDS and DNS cell names.

Cell-Relative Naming in a Standalone Cell

In addition to their global names, all CDS entries have a cell-relative, or local, name that is meaningful and usable only from within the local cell where that entry exists. The local name is a shortened form of a global name, and thus is a more convenient way to refer to resources within a user's own cell. Local names have the following characteristics:

- They do not include a global cell name.
- They begin with the /**.:** prefix.

Local names do not include a global cell name because the /**.:** prefix indicates that the name being referred to is within the local cell. When CDS encounters a /**.:** prefix on a name, it automatically replaces the prefix with the local cell's name, forming the global name. CDS can handle both global and local names, but it is more convenient to use the local name when referring to a name in the local cell.

The naming conventions required for the interaction of local and global directory services may at first seem confusing. In an environment where references to names outside of the local cell are necessary, the following simple guidelines can help make the conventions easy to remember and use:

- Know your cell name.
- Know whether a name that you are referring to is in your cell.
- When using a name that is within your cell, you can omit the cell name and include the /**.:** prefix.
- When using a name that is outside of your cell, enter its global syntax, including the /**...** prefix and the cell name.

- When someone asks for the name of a resource in your cell, give its global name, including the **/...** prefix.
- When storing a name in persistent storage (for example, in a shell script), use its global name, including the **/...** prefix. Local names (that is, names with a **/.:** prefix) are intended only for interactive use and should not be stored. (If a local name is referenced from within a foreign cell, the **/.:** prefix is resolved to the name of the foreign cell and the resulting name lookup either fails or produces the wrong name.)

Local File Names

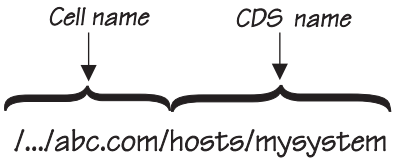
When referring to path names of files in the local cell, you can shorten a local name even further by using the **/:** prefix. This prefix translates to the root of the cell file system. The default name of the file system root is **/.:/fs**, which is one level down from the root of the cell namespace.

Examples — Determining Naming Environments

- The following example shows a global name for a price database server named in CDS. The server is used by the Portland sales branch of XYZ Company, an organization in the United States.

  As the example illustrates, global names for entries created in CDS look slightly different from pure GDS-style names. The first portion of the name, **/.../C=US/O=XYZ/OU=Portland**, is a global cell name that exists in GDS. The remaining portion of the name **/subsys/PriceMax/price_server1**, is a CDS name.

- The next example shows the global name of a DCE host at ABC Corporation. The global name of the company's cell, **/.../abc.com**, exists in DNS.



- The following example shows the global name for an entry created in GDS. The name represents user Ellie Bloggs, who works in the administrative organization unit of the Widget organization, a British corporation.

  ```
  /.../C=GB/O=Widget/OU=Admin/CN=Ellie Bloggs
  ```

  The GDS name syntax consists of a global prefix **/...** and a set of elements called Relative Distinguished Names (RDNs). Each RDN consists of one or more pairs of parts separated by an equals sign (=). The items that are separated by an equals sign are multiple AVAs (Attribute Value Assertions). The first part of a pair is an abbreviation that indicates the type of information. Some common abbreviations are Country (**C**), Organization (**O**), Organization Unit (**OU**), and Common Name (**CN**). The second part of the pair is a value.

  The cell name exists because cells must have names to be accessible in the global naming environment. The GDA looks up the cell name in the process of helping CDS in one cell find a name in another cell. Cell names are established during initial configuration of the DCE components. Before configuring a cell that will participate in standard intercell communication (that is, through the

DNS or GDS global directory services), the DCE administrator must obtain a unique cell name from either of the global naming environments, depending on whether the cell needs to be accessed through GDS or DNS.

- The following example shows the global cell names of two child cells.

```
                 Global Cell Name for Sales1
 ----------------------^--------------------------
/
          Parent Global Cell Name          Child Cell
 -------------^-------------------------   --^--
/                                         /      |
/.../C=US/O=XYZ/OU=Portland/subsys/PriceMax/Sales1

                Global Cell Name for Marketing
 ------------------------------------------------
/
          Parent Global Cell Name           Child Cell
 -------------^-------------------------   ---^--
/                                         /       |
  /.../C=US/O=XYZ/OU=Portland/subsys/PriceMax/Marketing
```

  The global cell name for each child includes:

  – The parent's global name, /**.../C=US/O=XYZ/OU=Portland/subsys/PriceMax**.

  – The child's unique CDS name, /**Sales1** or /**Marketing**.

- The following two names are equally valid when used within the cell named /**.../C=US/O=XYZ/OU=Portland**:

```
/.../C=US/O=XYZ/OU=Portland/subsys/PriceMax/price_server1
/.:/subsys/PriceMax/price_server1
```

- The child cell (**eng**) is named relative to its parent cell:

```
/.:/eng
```

  This type of naming allows you to access names in a child cell (for example, /**.:/eng/hosts/admin**) from the parent cell, without having to specify the global name of the cell.

- The following are all valid ways to refer to the same file from within the /**.../widget.com** cell:

```
/.../widget.com/fs/smith/myfile

/.:/fs/smith/myfile

/:/smith/myfile
```

Controlling Access to CDS Clerk and Server Management Operations

Introduction — Controlling Access to Management Operations

CDS authorization allows you to control the use of CDS commands that involve local management operations on CDS clerks and servers.

More Detailed Information — Controlling Access to Management Operations

Principal names for each clerk and server are stored in the security namespace. An object entry that contains the binding information for each clerk and server is stored in the CDS namespace in the /**.:/hosts** subdirectory. Servers are represented as /**.:/hosts**/*hostname*/**cds-server**. Clerks are represented as /**.:/hosts**/*hostname*/**cds-clerk**. Each clerk and server maintains a separate access control list that contains ACL entries specifying the principals allowed to perform these operations. Unlike the ACLs that are associated with names in the namespace, the ACLs that are associated with clerks and servers exist exclusively to provide local control of the use of these commands.

Whenever a new clerk or server is initialized, an access control list is created on the clerk or server system. An initial ACL entry is also created, granting the machine principal and the namespace authorization group (**subsys**/**dce**/**cds-admin**) read, write, and control permissions to the clerk or server process on that system. All other principals, both authenticated and unauthenticated, are granted read permission. The creation of this ACL entry ensures that, immediately after its creation, any user logged into the system as the machine principal is permitted to execute privileged clerk or server CDS commands.

Note: Use of the machine principal for this purpose is provided as a convenience and assumes that the account itself (user name and password) is already moderately secure. Namespace administrators may prefer to modify this scheme and grant permission to particular clerks and servers on behalf of other individual principals or authorization groups.

To edit an ACL that is associated with a CDS clerk or server, use the **dcecp** program's **acl modify** command with the **–change** option.

Note: Keep in mind that clerks and servers are also represented by entries in the namespace. To edit an ACL that is associated with the namespace entry for a CDS clerk or server, you must include the **–e** option, as well as the **–change** option, in the **acl modify** command line. For detailed instructions on how to modify an ACL on the CDS entry for a DCE resource, see “Editing ACLs on CDS Names” on page 5-30.

Example — Controlling Access to Management Operations

To change the permissions for the user **michaels** in the ACL that is associated with the CDS clerk on node **orion**, enter the following command:

```
dcecp> acl modify /.:/hosts/orion/cds-clerk -change {user michaels rw}
```

Creating Directories

Introduction — Creating Directories

By creating directories, you make it possible to replicate and manage groups of object entries according to where, how often, or by whom they are used. Grouping related object entries into separate directories also makes it easier to control access because it allows you to take advantage of default ACL entry propagation.

Note: Optionally, you can use non-PCS characters to name directories and files. To enable the multibyte CDS names code, set the internationalization environment variable first. The CDS program must be started in a session that is using a multibyte code page. Starting CDS in a single byte code page and then switching to a multibyte code page will not work as the locale is set only when the CDS daemons are started. In CDS you can use double-byte characters for directories and files.

More Detailed Information — Creating Directories

CDS cell configuration creates an initial hierarchy of directories under the root to provide DCE components with fixed locations within the namespace where they can create and catalog their object entries. Among the directories created by cell configuration is the **subsys** directory, beneath which Independent Software Vendors (ISVs) can create their own directories to store the object entries used by their distributed applications.

Alternatively, ISVs and other users of the namespace may prefer to create a hierarchy of directories of their own design under the root to store their information.

See *DSS for AIX Up and Running!* for more information on the initial hierarchy established by cell configuration.

Entering the directory create Command

Use the **directory create** command to create a new directory (master replica) with the name that you specify. When you use this command, by default CDS stores the master replica of the new directory in the same clearinghouse that stores the master replica of the new directory's parent directory.

Note: For the directory creation to succeed, the master replica of the new directory's parent directory must be available when you enter the command.

You can use the **directory create** command's **–clearinghouse** option to store the master replica of a new directory in a different clearinghouse than the parent directory's clearinghouse.

See the *DCE for AIX Administration Command Reference* for complete information on arguments and options for the **directory create** command.

Checking the ACL Entries for a New Directory

After you create a directory, verify that the users and applications for whom the directory was created have the appropriate permissions.

See the *DCE for AIX Administration Command Reference* for complete information on arguments and options for the **acl modify** command.

Upgrading the Directory Version on the Cell Root Directory

Upgrading the directory version on the cell root directory has special significance. This procedure implies that all CDS servers in the cell have been upgraded to the latest version, given that a cell root directory is replicated in all CDS servers in the cell. After you have set the **CDS_UpgradeTo** attribute on the cell root directory, the server software soon recognizes this and sets the **CDS_UpgradeTo** attribute on all directories in the cell. Eventually, the **CDS_DirectoryVersion** attribute on all affected directories in the cell will be upgraded to the new value.

Upgrading the Directory Version on a Directory

To use new features in a given release of CDS, you may need to explicitly update the directory version of a directory. This typically occurs when the servers replicating the directory all have been upgraded to the latest version of software, as older versions will not recognize the new features.

To upgrade the directory version, you need write permission to the directory and you must use the following commands:

```
dcecp> directory modify directory-name -add {CDS_UpgradeTo v.n} -single
dcecp> directory synchronize directory-name
```

Eventually all clearinghouses that contain a replica of this directory will detect the presence of the **CDS_UpgradeTo** attribute and upgrade the **CDS_ReplicaVersion** attribute on the appropriate replica. You can also use the following command on all clearinghouses that are replicating the directory:

```
dcecp> clearinghouse verify clearinghouse-name
```

This command forces the server background thread to run, thereby freeing you to perform other tasks until the job finishes.

Note: You cannot use the **clearinghouse verify** command on a remote host. This command is valid for the local host only.

After you have verified all affected clearinghouses, you will need to perform another skulk of the directory to finally set the **CDS_DirectoryVersion** attribute to the appropriate value. The **CDS_DirectoryVersion** attribute is not upgraded until all of the **CDS_ReplicaVersion** attribute values of all replicas contain the new value.

Permissions Required — Creating Directories

To create a directory, you need:

- Insert permission to the parent of the new directory,
- Write permission to the clearinghouse that stores the master replica of the new directory, and
- Read and insert permissions to the parent directory of the new directory for the server principal of the server system where you enter the **dcecp** program's **directory create** command.

  If the server is included in the server authorization group **subsys/dce/cds-servers**, these permissions should already be in place. If in doubt, use the **dcecp** program's **acl show** command on the parent directory to verify that the server principal has the appropriate permissions. See the *DCE for AIX Administration Command Reference* for complete information on the **acl show** command.

Examples — Creating Directories

- Use the following command to create a directory named /**.:/sales** and store the master replica of the new directory in the root directory's initial clearinghouse.

  ```
  dcecp> directory create /.:/sales
  ```

- To place the new directory created in the previous example into another clearinghouse (/**.:/Chicago1_CH**), enter the following command:

  ```
  dcecp> directory create /.:/sales -clearinghouse /.:/Chicago1_CH
  ```

- To verify user and application permissions, use the **acl show** command on the directory to see the associated ACL entries, as in the following example:

  ```
  dcecp> acl show /.:/sales
  {unathenticated r--t-}
  {group subsys/dce/cds-admin rwdtc}
  {group subsys/dce/cds-server rwdtc}
  {any_other r--t-}
  ```

Creating a Read-Only Replica

Introduction — Creating a Read-Only Replica

From time to time you will want to create read-only replicas of directories. You create read-only replicas of a directory to:

- Distribute the information that is contained in the directory throughout your network and to make the information more accessible to users and applications at other locations.
- Improve response time, especially in a namespace where users are dispersed over long distances. You should create read-only replicas in clearinghouses that are located near the user groups and applications that most frequently use the information that is contained in the directory.
- Preserve a backup of the information that is contained in the master replica of the directory. Maintaining multiple replicas ensures that the temporary loss of an individual replica does not cause an interruption in service and that the loss of a replica can be easily recovered. Even directories that store information used at only one particular site should be replicated in at least one other clearinghouse, preferably on a server at another location, so that a local failure at one site does not cause both replicas to be unreachable at the same time. See “Managing CDS Directories” on page 1-30 for more information on using directory replication as a means of backing up CDS information.

Read-only replicas of directories are safe from alteration by users. Users can look up information in a read-only replica, but they are not permitted to create new information or modify existing information.

Create read-only replicas with the **–replica** option of the **directory create** command. You should create the replicas in clearinghouses whose users need to access the directory but do not need, or are not permitted, to update its contents.

Steps — Creating a Read-Only Replica

Before you try to create a replica, verify that the clearinghouse containing the master replica of the directory you intend to replicate is running and reachable. To verify that this condition is satisfied, follow these steps:

1. For the directory that you intend to replicate, use the **directory show** command to display the directory's attribute values and look at the **CDS_Replicas** attribute. The value of this attribute shows the names of the clearinghouses that currently store a replica of the directory.
2. With this information, use the **directory show** command with the **–clearinghouse** and **–replica** options to verify that you can get a response from the clearinghouse that stores the master replica.

Note: If any read-only replicas in the directory's existing replica set are unavailable, the replication cannot complete. The normal skulking process completes the replication as soon as all replicas in the directory's replica set become available.

Permissions Required — Creating a Read-Only Replica

To create a replica, you need:

- Administer permission to the directory that you intend to replicate,
- Write permission to the clearinghouse that stores the new replica, and
- For the replica creation to succeed, the server principal for the server system where you enter the **directory create** command with the **–replica** and **–clearinghouse** options must have read, write, and administer permission to the directory that you intend to replicate.

  If the server is included in the server authorization group **subsys/dce/cds-servers**, these permissions should already be in place. If in doubt, use the **acl check** command to verify that the server principal

has the appropriate permissions. See the *DCE for AIX Administration Command Reference* for complete information on using the **acl check** command.

Examples — Creating a Read-Only Replica

- The following example displays the directory attribute values:

```
dcecp> directory show  /.:/sales
{RPC_ClassVersion {01 00}}
{CDS_CTS 1994-08-12-09:52:30.396-04:00I0.000/00-00-c0-f7-de-56}
{CDS_UTS 1994-08-12-09:52:31.506-04:00I0.000/00-00-c0-f7-de-56}
{CDS_ObjectUUID a37d84d0-b5dc-11cd-8ffe-0000c0f7de56}
{CDS_Replicas
 {{CH_UUID ce7ed810-b5db-11cd-8ffe-0000c0f7de56}
  {CH_Name /.../Chicago1/Chicago1_CH}
  {Replica_Type Master}
  {Tower {ncacn_ip_tcp 130.105.5.16}}
  {Tower {ncadg_ip_udp 130.105.5.16}}}}
{CDS_AllUpTo 1994-08-12-09:52:31.566-04:00I0.000/00-00-c0-f7-de-56}
{CDS_Convergence medium}
{CDS_ParentPointer
 {{Parent_UUID d034bc25-b5db-11cd-8ffe-0000c0f7de56}
  {Timeout
   {expiration 1994-08-12-09:52:30.396}
   {extension +1-00:00:00.000I0.000}}
  {myname /.../Chicago1/sales}}}
{CDS_DirectoryVersion 3.0}
{CDS_ReplicaState on}
{CDS_ReplicaType Master}
{CDS_LastSkulk 1994-08-12-09:52:31.566-04:00I0.000/00-00-c0-f7-de-56}
{CDS_LastUpdate 1994-08-12-09:52:31.506-04:00I0.000/00-00-c0-f7-de-56}
{CDS_RingPointer ce7ed810-b5db-11cd-8ffe-0000c0f7de56}
{CDS_Epoch a3df2a50-b5dc-11cd-8ffe-0000c0f7de56}
{CDS_ReplicaVersion 3.0}
```

- The following example verifies that you can get a response from the clearinghouse:

```
dcecp> directory show  /.:/sales  -replica -clearinghouse  /.:/Chicago1_CH
{RPC_ClassVersion {01 00}}
{CDS_CTS 1994-08-12-09:52:30.396-04:00I0.000/00-00-c0-f7-de-56}
{CDS_UTS 1994-08-12-09:52:31.506-04:00I0.000/00-00-c0-f7-de-56}
{CDS_ObjectUUID a37d84d0-b5dc-11cd-8ffe-0000c0f7de56}
{CDS_Replicas
 {{CH_UUID ce7ed810-b5db-11cd-8ffe-0000c0f7de56}
  {CH_Name /.../Chicago1/Chicago1_CH}
  {Replica_Type Master}
  {Tower {ncacn_ip_tcp 130.105.5.16}}
  {Tower {ncadg_ip_udp 130.105.5.16}}}}
{CDS_AllUpTo 1994-08-12-09:52:31.566-04:00I0.000/00-00-c0-f7-de-56}
{CDS_Convergence medium}
{CDS_ParentPointer
 {{Parent_UUID d034bc25-b5db-11cd-8ffe-0000c0f7de56}
  {Timeout
   {expiration 1994-08-12-09:52:30.396}
   {extension +1-00:00:00.000I0.000}}
  {myname /.../Chicago1/sales}}}
{CDS_DirectoryVersion 3.0}
{CDS_ReplicaState on}
{CDS_ReplicaType Master}
{CDS_LastSkulk 1994-08-12-09:52:31.566-04:00I0.000/00-00-c0-f7-de-56}
{CDS_LastUpdate 1994-08-12-09:52:31.506-04:00I0.000/00-00-c0-f7-de-56}
{CDS_RingPointer ce7ed810-b5db-11cd-8ffe-0000c0f7de56}
{CDS_Epoch a3df2a50-b5dc-11cd-8ffe-0000c0f7de56}
{CDS_ReplicaVersion 3.0}
```

DCE Permissions Supported by CDS

Introduction — DCE Permissions Supported by CDS

CDS supports the following DCE permissions:

- read (**r**)
- write (**w**)
- insert (**i**)
- delete (**d**)
- test (**t**)
- control (**c**)
- administer (**a**)

More Detailed Information — DCE Permissions Supported by CDS

Each permission has a slightly different meaning depending on the kind of CDS name with which it is associated. In general, the permissions are defined as follows:

- **Read Permission** — Allows a principal to look up a name and view the attribute values associated with it.
- **Write Permission** — Allows a principal to change the modifiable attributes associated with a name, except its ACLs.
- **Insert Permission** — Allows a principal to create new names in a directory (for use with directory entries only).
- **Delete Permission** — Allows a principal to delete a name from the namespace.
- **Test Permission** — Allows a principal to test whether an attribute of a name has a particular value without being able to actually see any of the values (that is, without having read permission to the name).

  Test permission provides application programs with a more efficient way to verify a CDS attribute value. Rather than reading an entire set of values, an application can test for the presence of a particular value.
- **Control Permission** — Allows a principal to modify the ACL entries that are associated with a name. (Note that read permission is also necessary for modifying the CDS entry ACLs; otherwise, **dcecp** and **acl_edit** cannot bind to the entry.) Control permission is automatically granted to the creator of a CDS name.
- **Administer Permission** — Allows a principal to issue CDS commands that control the replication of directories. Administer permission is for use with directory entries only.

A principal needs some permission to a name before it can try to perform management operations on the name. Otherwise, CDS does not recognize the name when the principal tries the management operation and returns an error that the name does not exist. If the principal has some permissions, but not those required to perform the operation, CDS returns an error explaining that the principal had insufficient rights to perform the operation.

The creator of a name is automatically granted all permissions that are appropriate for the type of name that is created. For example, a principal that is creating an object entry is granted read, write, delete, test, and control permissions to the object entry. A principal that is creating a directory is granted read, write, insert, delete, test, control, and administer permissions to the directory.

Note: Unlike the security mechanisms that are enforced by most other file directory systems, CDS does not require a principal to have access to all intermediate elements in the path name (full name) of a name in order to perform an operation on the name. For example, consider an object entry **object1** stored in the **/.:/sales** directory. In CDS, you can grant a principal access to the object entry **/.:/sales/object1** without necessarily granting the principal access to either the **/.:/sales** directory or the cell root directory (**/.:**).

Deleting a Clearinghouse

Introduction — Deleting a Clearinghouse

You may need to delete a clearinghouse from the server system on which it resides when:

- The system is scheduled for reallocation or removal from your network.
- You no longer want the system to function as a CDS server.

More Detailed Information — Deleting a Clearinghouse

Before You Delete a Clearinghouse

Before you attempt to delete a clearinghouse, verify the following:

- The clearinghouse is known to the server.
- The clearinghouse does not store a master replica.

When you clear a clearinghouse, the server on which the clearinghouse was running no longer has information about the clearinghouse in its internal memory. If you try to delete the clearinghouse, CDS cannot find it and returns a message that it does not exist. Before you can delete a cleared clearinghouse, you must recreate it using the **clearinghouse create** command.

CDS does not allow you to delete a clearinghouse that contains a directory's master replica. Before you delete such a clearinghouse, you must designate another replica in that directory's replica set as the master replica. If no other replicas of the directory exist, create a read-only replica at another clearinghouse and then designate it as the directory's new master replica before you delete the original master replica from the clearinghouse. See “Modifying a Directory's Replica Set” on page 5-50 for additional information.

To Delete a Clearinghouse

Use the **clearinghouse delete** command to delete a clearinghouse. The command also deletes the clearinghouse's associated clearinghouse object entry and all read-only replicas from the clearinghouse.

Use the **clearinghouse delete** command to delete the clearinghouse. The command also deletes the clearinghouse's associated clearinghouse object entry and all read-only replicas from the clearinghouse.

Clearinghouse deletion can take some time to complete. CDS deletes a clearinghouse only after successfully completing a skulk of all directories that stored read-only replicas in the clearinghouse.

The following command deletes the **/.:/Paris2_CH** clearinghouse:

```
dcecp> clearinghouse delete /.:/Paris2_CH
```

Permissions Required — Deleting a Clearinghouse

The following permissions are required for deleting a clearinghouse:

- You need write and delete permissions to the clearinghouse and administer permission to all of the directories that store replicas in the clearinghouse.
- The server principal needs delete permission to the associated clearinghouse object entry and administer permission to all directories that store replicas in the clearinghouse.

Deleting Directories

Introduction — Deleting Directories

You may sometimes want to delete a directory from your namespace when the information that it contains is no longer needed by users. You must take two considerations into account when deleting a directory:

- **Does the directory contain child directories or the entries for any other CDS object?** Before a directory may be deleted, it must be empty.
- **Are there any replicas of the directory?** They must each be deleted separately.

Both of these factors are discussed in the following sections.

More Detailed Information — Deleting Directories

Deleting a Non-Replicated Directory

To delete a directory that has no replicas, use the **dcecp** utility's **directory delete** command (see "Examples — Deleting Directories" on page 5-23). For example, to delete the directory **/.:/sales**, all of its immediate contents, and the contents of any of its child directories, you would enter:

```
dcecp> directory delete  /.:/sales -tree
```

Note: Be careful when using the **–tree** option of the **directory delete** command. The command does not ask you to confirm that you want to delete the directory that you specify on the command line; it proceeds immediately with the delete operation. This can result in the loss of directories that you want to keep.

Remember that you can change the behavior of **dcecp** commands through scripts. In the case of the **directory delete** command, you could write a script that prompted for a confirmation of the delete operation whenever the command was run with its **–tree** option. See "Writing Scripts and **dcecp** Objects" on page 1-25 for a discussion of writing scripts.

A way to guard against the inadvertent deletion of directories and their entries is to view the contents before you run the **directory delete** command. To display the contents of a CDS directory by entry type, use the **directory list** command with the **–object**, **–link**, and **–directory** options.

If a directory to be deleted is not empty, the **directory delete** command will fail. To recover from this kind of failure, you must remove all the entries in the directory and its child directories, then run the **directory delete** command again. Use the **link delete** and **object delete** commands to delete the soft links and object entries in any directories. Then run the **directory delete** command to delete the directories.

Deleting a Directory Replica

If a directory is replicated, all the replicas have to be deleted individually. Then the directory can be deleted using the commands described in the previous section.

To display a list of all replicas of a directory, use the **dcecp** utility's **directory show** command (see "Examples — Deleting Directories"). Look at the **CDS_Replicas** attribute of the directory in the list. Each replica's **CDS_Replicas** attribute has several subattributes. Look at the **CH_Name** sub-attribute for each replica to get the name of the clearinghouse where it is located.

The name of the directory and the name of the clearinghouse can be used to uniquely identify each replica. Use these names in a series of **directory delete** commands to remove the replicas. The name of each replica is the argument to the command and the name of the clearinghouse should be used as the value of the **–clearinghouse** option. The **–replica** option should also appear in the command line to indicate that the directory to be deleted is a replica.

Note: The **directory delete** command does not require that directory replicas are empty in order to operate on them. It will delete the replicas, all their contents, and their child directories immediately, without prompting for confirmation of the operation.

You may want to write a **dcecp** script that looks at the **CDS_Replicas** attribute, finds all the replicas and deletes them with one command. See "Writing Scripts and **dcecp** Objects" on page 1-25 for a discussion of writing scripts.

Permissions Required — Deleting Directories

To delete a directory, you need:

- Delete permission to the directory,
- Write permission to the clearinghouse that stores the master replica of the directory, and
- Administer permission to the parent directory or delete permission to the child pointer that points to the directory you intend to delete for the server principal of the server from which you enter the **directory delete** command.

  If the server is included in the server authorization group **subsys/dce/cds-servers**, these permissions should already be in place. If in doubt, invoke the **dcecp** utility **acl show** and verify that the server principal has the appropriate permissions. See *DCE for AIX Administration Command Reference* for complete information on using the **acl show** command.

Examples — Deleting Directories

- The following is an example in which a directory named /**.:/sales** is deleted. The directory has one object entry and one soft link:

```
dcecp>  directory list /.:/sales -simplename
work_disk link1
dcecp> directory list /.:/sales -simplename -object
work_disk
dcecp> directory list /.:/sales -simplename -link link1
dcecp> directory delete /.:/sales -tree
dcecp> directory show /.:/sales
Error: Requested entry does not exist
```

- To display a list of all replicas of a directory, use the **dcecp** utility's **directory show** command, as follows:

```
dcecp> directory show /.:/sales
{RPC_ClassVersion {01 00}}
{CDS_CTS 1994-05-06-11:41:05.314-05:00I0.000/08-00-09-25-13-52}
{CDS_UTS 1994-06-21-03:06:08.842-05:00I0.000/08-00-09-25-13-52}
{CDS_ObjectUUID 5f97a584-bf9b-11cd-9362-080009251352}
{CDS_Replicas
 {{CH_UUID de3401e6-bb98-11cd-aac5-080009251352}
  {CH_Name /.../absolut_cell/absolut_ch}
  {Replica_Type Master}
  {Tower {ncacn_ip_tcp 130.105.5.93}}
  {Tower {ncadg_ip_udp 130.105.5.93}}}}
{CDS_AllUpTo 23854-01-29-19:45:44.841-05:00I0.000/08-00-09-25-13-52}
{CDS_Convergence medium}
{CDS_ParentPointer
 {{Parent_UUID df13b228-bb98-11cd-aac5-080009251352}
  {Timeout
   {expiration 1994-08-24-19:30:30.827}
   {extension +1-00:00:00.000I0.000}}
  {myname /.../absolut_cell/sales}}}
{CDS_DirectoryVersion 3.0}
{CDS_ReplicaState on}
{CDS_ReplicaType Master}
{CDS_LastSkulk 1994-01-29-19:45:44.841-05:00I0.000/08-00-09-25-13-52}
{CDS_LastUpdate 1994-06-21-03:06:08.842-05:00I0.000/08-00-09-25-13-52}
{CDS_Epoch 60ac0730-bf9b-11cd-9362-080009251352}
{CDS_ReplicaVersion 3.0}
```

- The following example shows the **directory delete** command with the **–replica** option:

```
dcecp> directory delete  /.:/sales -replica -clearinghouse /.:/NY1_CH
```

Deleting Read-Only Replicas

Introduction — Deleting Read-Only Replicas

Sometimes you may need to delete a read-only replica when the information that it contains is no longer needed by the local users of the clearinghouse in which the replica is stored. You may also need to delete a read-only replica to prepare for deleting the directory of which the replica is a member or before permanently removing the clearinghouse in which the replica is stored.

More Detailed Information — Deleting Read-Only Replicas

Use the **directory delete** command with the **–replica** and **–clearinghouse** options to delete a replica from the clearinghouse that you specify. For example, the following command deletes a replica of the /**.:/eng** directory from the /**.:/Chicago2_CH** clearinghouse:

```
dcecp> directory delete /.:/eng -replica -clearinghouse /.:/Chicago2_CH
```

Note: You can delete a directory's master replica only by deleting the directory itself (by using the **directory delete** command). See “Managing CDS Directories” on page 1-30 for complete information on how to delete a master replica.

Permissions Required — Deleting Read-Only Replicas

To delete a replica, you need:

- Administer permission to the directory whose replica you want to delete, and
- Write permission to the clearinghouse from which you are deleting the replica.

Directory Services Usage by DCE Components

The DCE Directory Service is a fundamental service that applications can rely on and use to their advantage. This section describes how other DCE components use the Directory Service.

The DCE Remote Procedure Call (RPC) interface facilitates the development and use of distributed applications that follow a client/server model. In the RPC model, *clients* are programs that make remote procedure calls and *servers* are programs that execute the procedures. The DCE RPC software stores information in the Directory Service about the addresses of RPC servers and the interfaces they support.

When an RPC client wants to make a call to a particular server, it can query the Directory Service for the information necessary to contact that server. If the client wants to access a specific resource that is named in the Directory Service, it can query for that specific name. If a client application knows the type of service that it wants (such as C compilers, printers, or employee information), but does not know the address of a specific server, it can also use the Directory Service to find that information.

The DCE Security Service, which verifies the identity of users when they log in, uses the Directory Service to store the addresses of its authentication servers.

The DCE Distributed File Service (DFS) provides a location service for filesets (logical groups of files) so that users can access remote files as if they are on the local system. DFS uses the Directory Service to determine how to contact its fileset location servers.

The DCE Distributed Time Service (DTS) is responsible for synchronizing system clocks in the network. Synchronized clocks are important to any distributed application that needs to keep track of the order in which events occur across multiple systems. DTS uses the Directory Service to find out how to locate its time servers.

Directory Services and the Cell Environment

Introduction — Directory Services and the Cell Environment

Although end users may not recognize the distinction, the DCE naming components that operate within a cell and outside a cell are different. This section introduces the following main components of the DCE naming environment and explains their relationship to the cell:

- Cell Directory Service (CDS)
- Domain Name System (DNS)
- Global Directory Agent (GDA)
- Global Directory Service (GDS)

Note: DCE for AIX does not provide the GDS function. However, if another software product provides this function in the DCE cell, DCE for AIX can use this function to locate other cells.

More Detailed Information — Directory Services and the Cell Environment

CDS is a high-performance distributed service that provides a consistent, location-independent method for naming and using resources inside a cell (intracell). CDS also provides communication between cells (intercell) when cells are connected into a hierarchy.

GDS supports the global naming environment between cells (intercell) and outside of cells (extracell). GDS is an implementation of a directory service standard known as X.500. This standard is specified by the International Organization for Standardization (ISO) 9594 and the International Telegraph and Telephone Consultative Committee (CCITT) X.500 series. Because it is based on a worldwide standard, GDS offers the opportunity for a universally interoperable global directory.

Figure 5-5 represents a hypothetical configuration of two cells that each use GDS to access names in the other cell. Names that are stored directly in GDS also are accessible from each cell. CDS is the directory service within each cell. The same organization administers both cells, which are configured based on geographic location and network topology.

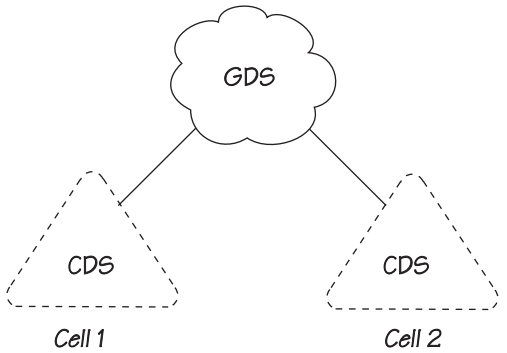


Figure 5-5. Cell and Global Naming Environments

DNS is a widely used existing global name service for which DCE offers support. Many networks currently use DNS primarily as a name service for Internet host names. Although DNS is not a part of the DCE technology offering, the DCE Directory Service contains support for cells to interoperate through DNS.

The GDA is the DCE component that makes cell interoperation possible. The GDA enables CDS to access a name in another cell through one of the global naming environments (GDS or DNS). The GDA is an independent process that can exist on a system separate from a CDS server (although, by default, the DCE configuration script configures the GDA on the same machine as a CDS server). CDS needs to be able to contact at least one GDA to participate in the global naming environment.

Figure 5-6 on page 5-27 shows how the GDA helps CDS access names outside a cell. When CDS determines that a name is not in its own cell, it passes the name to a GDA, which searches the appropriate naming environment (CDS, GDS, or DNS) for more information about the name. The GDA returns information that enables the original CDS server to contact the CDS server in whose cell the name resides. The GDA can help CDS find names in a cell that is registered in DNS (Scenario A) or a cell that is registered in GDS (Scenario B). The GDA decides which global service to use based on the syntax of the name.

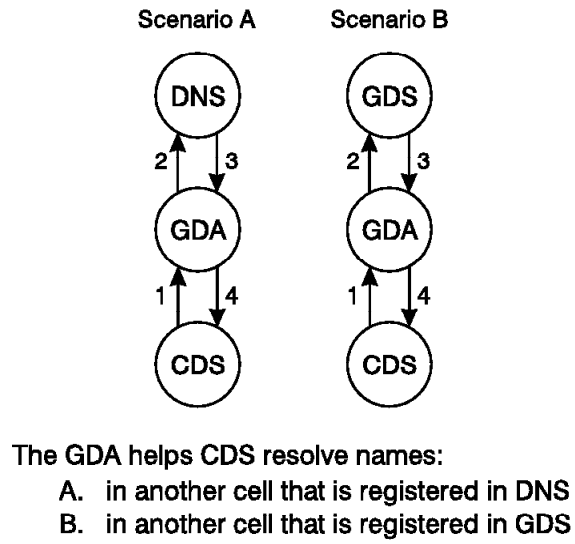


Figure 5-6. Interaction of CDS, GDA and Global Directory Services

Note: GDS is not supported in this release of DCE. It is, however, supported in DCE 1.3 on AIX 3.2.5. Therefore, to resolve a GDS cell name, an AIX 3.2.5 system running DCE 1.3 and GDS must exist in the cell. The GDA must also be on the same workstation as the GDS client.

Directory Service Usage

Introduction — Directory Service Usage

With the exception of DCE administrators, users normally access directory services indirectly, through an application interface. An application can interact with the DCE Directory Service on behalf of users who create a name for a resource and subsequently refer to it by that name.

Examples — Directory Service Usage

The following examples, both real and hypothetical, explain some of the ways that users can access the Directory Service:

- A user invokes a spell-checking application on a new document. The application contains DCE Remote Procedure Call (RPC) client code on the user's local system. The RPC client contacts the Directory Service for information on an available spell-checking server. The Directory Service returns the address of the server, the protocol type it uses to communicate, and a Universal Unique Identifier (UUID) that represents an interface. Using this information, the RPC client makes a remote call to the server and the server checks the spelling in the user's document. The user is unaware that implementing the spell checker involved a call to the Directory Service and interaction with a remote server.
- A user logging into a system enters a name and password. The Directory Service helps the login program locate an authentication server, which verifies the user's identity in an authentication database.
- A user enters a file specification. The Directory Service provides the address of a DFS fileset location database, which contains the network address of a server that allows the user to access the file.
- A user enters the name of a computer conference or electronic bulletin board and the Directory Service provides an address, allowing the application to connect to the conference service.

- By entering a name or some information about a printer's capabilities, a user can learn the printer's network address. For example, the user might want to find the address of the closest and fastest available color printer.
- A user needs information from an employee in the marketing department. The user remembers that the employee's last name is Wong, but cannot remember the first name. By entering the last name and department name in an employee locator application, the user can check the Directory Service for information on all Wongs in the marketing department and find out how to contact the employee.
- A user enters a report in a problem-tracking database. Although the database was recently moved to a new node, the user is not aware of the change because the database is always referred to by its name only. The Directory Service stores the current network address and provides it to the problem-tracking application and any other application that requests it.

Disabling Clerks and Servers

Introduction — Disabling Clerks and Servers

You may occasionally have to disable the clerk or server that is running on a particular system when you need to perform diagnostic or troubleshooting work that requires suspension of active clerk or server processes. You can generally use the configuration process to start and stop DCE daemons (see *DSS for AIX Up and Running!* for configuration information). You can disable CDS clerks and servers by using the **disable clerk** and **disable server** commands of **cdscp**.

Steps — Disabling Clerks and Servers

- To disable the clerk that is on the local node, enter the following command:

  ```
  cdscp> disable clerk
  ```

- To disable the server that is on the local node, enter the following command:

  ```
  cdscp> disable server
  ```

Displaying the Contents of a Clearinghouse

Use the **dcecp** program's **clearinghouse show** command with the **–clear** option and specify the **CDS_CHDirectories** attribute to display the directory names of all directories stored in a particular clearinghouse. For example, to display the names of the directories that are stored in the clearinghouse /**.:/Chicago2_CH**, enter the following command:

```
cdscp> clearinghouse show /.:/Chicago2_CH CDS_CHDirectories -clear
```

See “Listing the Contents of Directories” on page 5-37 for another example of displaying clearinghouse information.

Displaying the Namespace with the CDS Browser

Introduction — Displaying the Namespace

The CDS Browser is a tool for viewing the content and structure of a namespace. It runs on workstations with windowing software that is based on the OSF/Motif graphical user interface. The Browser can display the overall directory structure as well as the contents of directories, enabling you to monitor growth in the size and number of directories in your namespace. You can also customize the CDS Browser to display only a specific class of object names.

More Detailed Information — Displaying the Namespace

Displaying the Default Namespace

The CDS Browser lets you view the default namespace for your system. You can see only the entries in the namespace to which you have read permission. Directories to which you do not have read permission do not appear. When you use the CDS Browser, it sets the confidence level of clerk calls to **low**.

When you start the CDS Browser, an icon representing the root directory is the first item to display in the window. Directories, soft links, and object entries all have distinct icons associated with them. Figure 5-7 shows the Browser icons and what they represent.

Icon	Entry Type	Icon	Entry Type
	Directory		Object entry
	Clearinghouse object entry		Soft link

Figure 5-7. CDS Browser Icons and Their Meaning

To expand (open) the root directory, double-click on it. Double-click on the expanded directory to collapse (close) it. When you expand a directory, you see all of the soft links and object entries that it contains. Object entries can represent clearinghouses or any resource for which a client application creates entries in the namespace. Object entries representing clearinghouses are shown with a different icon than are ordinary object entries. All entries, such as object entries, soft links, and directories, are shown indented from their parent directories.

Expanding and Collapsing Selected Directories

By double-clicking on single directories, you can continue expanding a particular directory path name one level at a time. The following describes two other methods to expand all directories at the same time or to expand selected groups of directories.

- To expand or collapse a group of directories, select them and double-click on them. Since double-clicking has a toggle effect, you can expand or collapse groups of directories only one level at a time. If you double-click multiple directory levels at one time, the result might be the opposite of what you expect.
- To expand or collapse selected directories level by level, click on the first directory that you want to select, then continue selecting directories by shift-clicking (pressing <Shift> and clicking) on them.

When you select the last directory, press <Shift> and double-click, instead of single-clicking. This selects the last directory and expands or collapses all of the directories that you selected.

Expanding and Collapsing the Entire Cell Namespace

To expand all directories on all levels at the same time, choose the `Expand All` option from the `File` menu. Likewise, choose `Collapse All` from the `File` menu to close an expanded namespace.

Note: Use `Expand All` with care if you have a large namespace. The larger a namespace, the longer it takes to display its entire contents.

Filtering the Namespace Display

Using the `Filters` menu, you can selectively display object entries of a particular class. For example, to view the entries for clearinghouse objects only, choose the class **CDS_Clearinghouse** from the `Filters` menu. For any directory that you expand after choosing a filter, you see only names of objects whose class matches the filter. Notice that soft links are still displayed because they are not object entries and only object entries can be filtered out. To reset the filter so that you can again view all object entries, choose the asterisk (*) from the `Filters` menu.

Navigating the Namespace

When you begin expanding the namespace, it can exceed the boundaries of your CDS Browser window, even if you enlarge the window. You can use the horizontal and vertical scroll bars and stepping arrows to scroll through the namespace.

Dragging the slider up and down the vertical scroll bar on the right side of the display window produces an index window. The index window shows the name where the slider is currently positioned in the namespace. When the index window contains the name that you want to view, release the mouse button to position that name at the top of the CDS Browser window.

In displays that are larger than the length of the window, scrolling through directory levels can produce a reference line toward the top of the window. The line orients you by showing the full directory path name from the current name to the root. It also indicates that you have scrolled past other parts of the namespace that are no longer displayed.

Steps — Displaying the Namespace

- To start the CDS Browser, enter the following command at your system prompt:

  ```
  $ cdsbrowser
  ```

- To end a CDS Browser session and return to your system prompt, choose `Quit` from the `File` pull-down menu. See the *DCE for AIX Administration Command Reference* for a complete description of the **cdsbrowser** command.

Editing ACLs on CDS Names

Introduction — Editing ACLs on CDS Names

To edit an ACL that is associated with an entry in the CDS namespace for a child pointer, clearinghouse, soft link, or some other CDS object, specify the **–entry** option to any **dcecp** program **acl** command. The **–entry** option is especially useful in case of an ambiguous path name. In some cases, a path name can resolve to a leaf object in the DCE Directory Service and to an object in some other DCE component that supports ACLs. In these cases, you must use the **–entry** option to edit the leaf object in CDS. You do not need to specify this option to edit ACLs that are associated with actual clearinghouses or directories.

Examples — Editing ACLs on CDS Names

- To edit the permissions in the Object ACL that is associated with a CDS entry for a clearinghouse named /**.:/Paris1_CH**, you enter the following command:

  ```
  dcecp> acl modify /.:/Paris1_CH -entry -change {unauthenticated -}
  ```

- To edit the permissions in the Object ACL that is associated with the /**.:/Paris1_CH** clearinghouse itself, you enter the following command:

  ```
  dcecp> acl modify /.:/Paris1_CH -change {unauthenticated -}
  ```

- Another example is the soft link /.../**eng_printer**. The target of this soft link is /**.../boston.com/print_server**. To edit the soft link leaf entry that is in the CDS namespace, enter the following command:

  ```
  dcecp> acl modify /.../eng_printer -change -entry {group subsys/dce/cds-admin rwdtc}
  ```

Enabling Other Cells to Find Your Cell

Introduction — Enabling Other Cells to Find Your Cell

Note: Although DCE for AIX does not provide the Global Directory Service (GDS), information is provided for your reference if GDS is available in your cell from another product (such as DCE 1.3 for AIX).

The GDA is the mechanism that allows CDS clerks in your local cell to find other cells. To make your cell accessible to others, you must create an entry for it in one of the currently supported global naming environments. Before you do this, obtain a unique cell name from the appropriate naming authority. See the *DSS for AIX Up and Running!* for details.

After you configure a cell, name it, and initialize the cell namespace, you can use the *cdscp* program's **show cell** command to obtain data that you need to create or modify the cell entry in DNS or GDS. The data in a cell entry is what the GDA passes to CDS after looking up a cell name. CDS, in turn, uses the information to contact servers in the cell. The following sections describe how to define and maintain a cell entry in DNS or GDS. These sections assume a basic familiarity with DNS and GDS; for details, see the appropriate documentation for each global name service.

More Detailed Information — Enabling Other Cells to Find Your Cell

Defining a Cell in the Domain Name System

Names in DNS are associated with one or more data structures called *resource records*. The resource records are stored in a data file whose name and location are implementation specific. To create a cell entry, you must edit the data file and create two resource records for each CDS server that maintains a replica of the cell namespace root.

The first resource record, whose type can be AFSDB or MX, contains the DCE host name of the system where the CDS server resides. You can use MX as an alternative to using AFSDB. The second record of type TXT contains the following information about the replica of the root directory that the server maintains:

- The Universal Unique Identifier (UUID) of the cell namespace (in hexadecimal notation)
- The type of the replica (master or read-only)
- The global CDS name of the clearinghouse where the replica resides
- The UUID of the clearinghouse (in hexadecimal notation)
- The DNS name of the host where the clearinghouse resides

To create a new resource record in the DNS space, select the appropriate DNS data file and then cut and paste (or type) the **show cell** output into this file. Make sure that you copy to all new resource records the value of the **TXT =** string and the DNS name of the server where the clearinghouse resides.

If, over time, you create additional replicas of the root directory, move a clearinghouse, or make other changes that affect the cell resource records, you can follow the same procedure as described here.

See "Global and Cell Considerations" on page 1-4 for complete information on how to contact the NIC Domain Registrar to register a domain name.

Defining a Cell in the Global Directory Service

GDS is not supported in this release. See the DCE for AIX 1.3 documentation for instructions on adding cell binding information to a GDS object.

Examples — Enabling Other Cells to Find Your Cell

- The following example shows a set of AFSDB resource records for a cell that is named **cs.tech.edu**, in which two replicas of the root directory exist. Note that only the first resource record contains the cell name. The second, third, and fourth records are assumed to be associated with the same cell because they do not contain a cell name. The `TTL` heading stands for Time To Live, a value (in seconds) after which the data is no longer considered valid in a DNS cache (the value shown specifies a default value of 1 week). The `IN` class indicates that the protocol is Internet, and the subtype of `2` indicates that a name server exists on the host named in the record.

```
;First Replica:
;Name             TTL     Class     Type    Subtype     Host
cs.tech.edu       604800  IN        AFSDB    2          fox.cs.tech.edu
;Name             TTL     Class     Type    Rdata
                  604800  IN        TXT     (1          ;version
                  fd3328c4-2a4b-11ca-af85-09002b1c89bb  ;ns uuid
                  Master                                ;Replica1 type
                  /.../cs.tech.edu/cs1_ch               ;ch name
                  fd3328c5-2a4b-11ca-af85-09002b1c89bb  ;ch uuid
                  fox.cs.tech.edu                       ;host
;Second Replica:
                  604800  IN        AFSDB    2          rox.cs.tech.edu
                  604800  IN        TXT     (1          ;version
                  fd3328c4-2a4b-11ca-af85-09002b1c89bb  ;ns uuid
                  Read-only                             ;Replica2 type
                  /.../cs.tech.edu/cs2_ch               ;ch name
                  fd3429c4-2a4b-11ca-af87-09002b1c89bb  ;ch uuid
                  rox.cs.tech.edu                       ;host
```

- You can use MX records as an alternative to using AFSDB. The following example shows a set of MX resource records for the same cell, **cs.tech.edu**, in which two replicas of the root directory exist.

```
;First Replica:
;Name            TTL       Class     Type    Preference  Exchange
cs.tech.edu      604800    IN        MX        1         fox.cs.tech.edu.
;Name            TTL       Class     Type     Rdata
                 604800    IN        TXT      (1        ;version
                 fd3328c4-2a4b-11ca-af85-09002b1c89bb   ;ns uuid
                 Master                                 ;Replica1 type
                 /.../cs.tech.edu/cs1_ch                ;ch name
                 fd3328c5-2a4b-11ca-af85-09002b1c89bb   ;ch uuid
                 fox.cs.tech.edu)                       ;host
;Second Replica:
                 604800    IN         MX       2        rox.cs.tech.edu.
                 604800    IN        TXT      (1        ;version
                 fd3328c4-2a4b-11ca-af85-09002b1c89bb   ;ns uuid
                 Read-only                              ;Replica2 type
                 /.../cs.tech.edu/cs2_ch                ;ch name
                 fd3429c4-2a4b-11ca-af87-09002b1c89bb   ;ch uuid
                 rox.cs.tech.edu)                       ;host
```

- After you configure a cell, you can use the *cdscp* program's **show cell** command to display the information that is required to construct resource records like those shown in the previous example. The following is an example **show cell** command that displays the DNS-formatted output for a cell named /**.../cs.tech.edu**. Note that the **show cell** command does not display the DCE host name (including domain) at the end of the TXT record.

```
cdscp> show cell /.../cs.tech.edu as dns

                          SHOW
                          CELL /.../cs.tech.edu
                            AT   1994-09-18-17:15:00
                           TXT = 1 de86b5cd-75ea-11ca-bad8-08002
b1c8f1f Master /.../cs.tech.edu/NY1_ch dc730a9c-75ea-11ca-bad8-08002b1c8f1f

                                TXT = 1 de86b5cd-75ea-11ca-bad8-08002
b1c8f1f Readonly /.../cs.tech.edu/NY2_ch 391f15e8-75ef-11ca-a4f4-08002b1c8f1f
```

Error Handling

Introduction — Error Handling

Most of the errors that the **directory merge** command encounters during its operations are caused by the following:

- Duplicate names that are detected during the merge
- Names in the source subtree whose master clearinghouses were not reachable when the command was executing
- Entries not created in the target location due to insufficient permissions

More Detailed Information — Error Handling

Duplicate Names

If the full name of a CDS object entry or soft link is identical to a full name of an object entry or soft link at the target location, the **directory merge** command lists these duplicate names and stops. Duplicate names are not merged to avoid overwriting and destroying the identical names in the target directory.

If duplicate names exist, you need to decide which names you want to preserve — the names in the source subtree or the names in the target subtree. When you have made your decision, proceed in the following manner:

1. Use the **dcecp** program's **create** commands to recreate (under a new name) any duplicate object entry or soft link as a new object entry or soft link in the source or target subtree. Then delete the duplicate name.
2. When you are certain that connectivity to the affected clearinghouses can be maintained, rerun the **directory merge** command to merge the contents of the source and target directories.

Unreachable Name Failures

Sometimes, the clearinghouse that stores the master replica of a directory you are trying to merge is disabled or unreachable when you enter the **directory merge** command. When this happens, the command cannot create the directory and the entries it contains at the new target location.

When unable to merge a name for this reason, the **directory merge** command displays an error message specifying the name that could not be created and terminates.

Permissions Required — Error Handling

The **directory merge** command cannot create CDS objects at a target location if it lacks the appropriate permissions. If the command returns error messages indicating insufficient permissions, you need to examine the ACLs for the target clearinghouse, directories, and object entries to see the current permissions and change the inappropriate ones. Table 5-1 shows the permissions required to create directories and other CDS object entries at the target.

Table 5-1. Permissions Required to Create Target Objects

Objects	Required Permissions
directory	• Write permission to the clearinghouse that is to store the master replica of the new directory • Insert and read permissions to the parent of the new directory • Insert and read permissions to the initial container for the new directory • The server principal also needs read and insert permissions to the parent directory of the new directory
Other CDS object	• Insert and read permissions to the directory where it is to be created • Insert and read permissions to the initial object for its object type

Forcing the Clearinghouse to Checkpoint to Disk

Under normal operations, the server will periodically checkpoint the clearinghouse from memory to disk. However, you can perform this task immediately by having write permission to the server and enter the **dcecp** program's **clearinghouse initiate** command with the **–checkpoint** option. For example, to checkpoint the clearinghouse /**.:Boston3_CH** from memory to disk, enter the following command:

```
dcecp> clearinghouse initiate /.:/Boston3_CH -checkpoint
```

Global Directory Agent Operation

Introduction — GDA Operation

The GDA is an intermediary between CDS clerks in the local cell and CDS servers in other cells. A CDS clerk treats the GDA like any other name server passing it name lookup requests. However, the GDA provides the clerk with only one specific service. It looks up a cell name in the GDS or DNS namespace and returns the results to the clerk. The clerk then uses those results to contact a CDS server in the foreign cell.

More Detailed Information — GDA Operation

A GDA must exist inside any cell that wants to communicate with other cells. It can be on the same system as a CDS server or it can exist independently on another system. You can configure more than one GDA in a cell for increased availability and reliability. Like a CDS server, a GDA is a principal and must authenticate itself to clerks.

CDS finds a GDA by reading address information that is stored in the **CDS_GDAPointers** attribute associated with the cell root directory. Whenever a GDA process starts, it creates a new entry or updates an existing entry in the **CDS_GDAPointers** attribute. The entry contains the address of the host on which the GDA is currently running. If multiple GDAs exist in a cell, they each create and maintain their own address information in the **CDS_GDAPointers** attribute.

When a CDS server receives a request for a name that is not in the local cell, the server examines the **CDS_GDAPointers** attribute of the cell root directory to find the location of one or more GDAs. Figure 5-8 shows how a CDS clerk and CDS server interact to find a GDA.

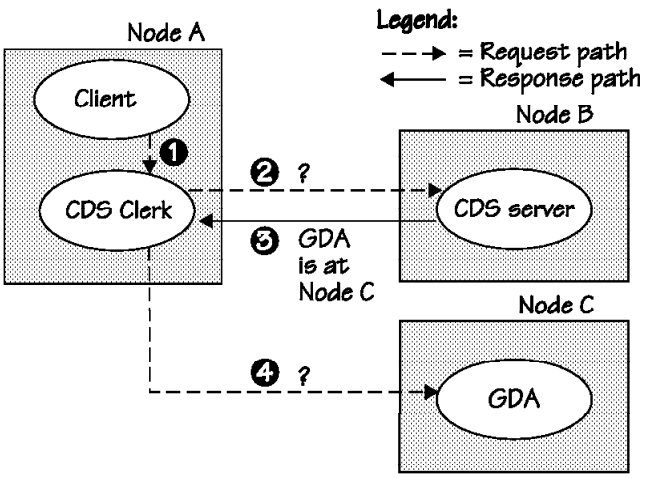


Figure 5-8. How the CDS Clerk Finds a GDA

- The following steps summarize the GDA search that is illustrated in Figure 5-8:
  1. On Node A, a client application passes a global name beginning with the /**...** prefix to the CDS clerk.
  2. The clerk passes the lookup request to a CDS server that it knows about on Node B.
  3. The server's clearinghouse contains a replica of the cell root directory, so the server reads the **CDS_GDAPointers** attribute and returns the address of Node C where a GDA is running.
  4. The clerk passes the lookup request to the GDA.

Figure 5-9 on page 5-36 shows how the CDS and a GDA interact to find a name in a foreign cell that is defined in DNS. Suppose the name is /**.../widget.com/printsrv1**, which represents a print server in the foreign cell.

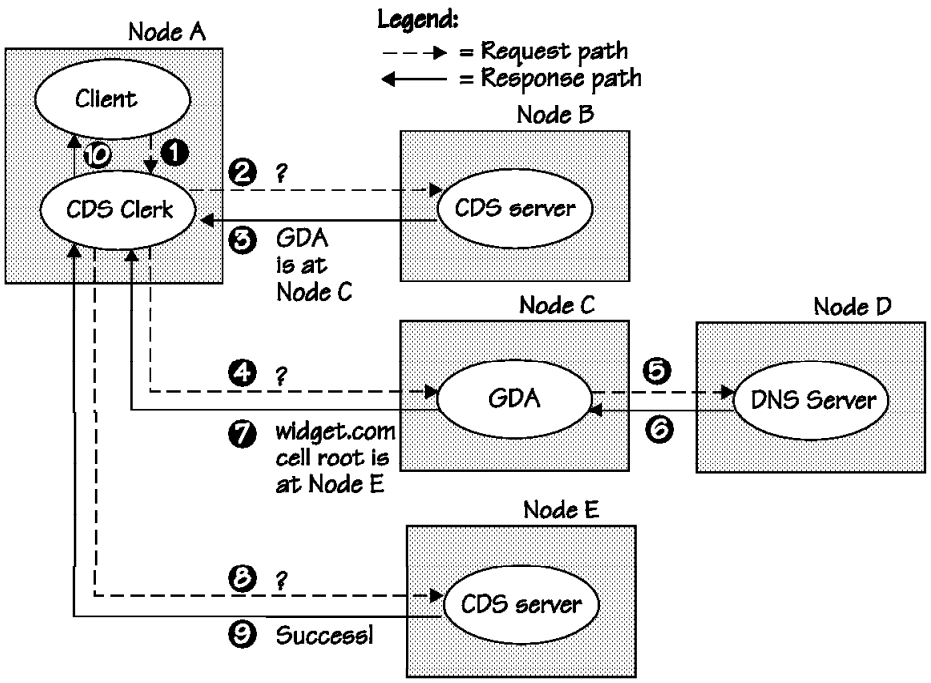


Figure 5-9. How the GDA Helps CDS Find a Name

- The following steps summarize the name search that is illustrated in the preceding figure:
  1. The client application passes the name /**.../widget.com/printsrv1** to the CDS clerk.
  2. The clerk passes a lookup request to a CDS server that it knows about on Node B.
  3. The server's clearinghouse contains a replica of the cell root directory, so the server looks up the **CDS_GDAPointers** attribute and returns the address of Node C where a GDA is running.
  4. The clerk passes the lookup request to the GDA.
  5. The GDA recognizes that the name is a DNS-style name, so it assumes the second component is a cell name that is defined in DNS. It passes that portion of the name (**widget.com**) to DNS. For simplicity, the figure shows only one DNS server. More than one DNS server can actually be involved in resolving a global cell name.

     Note: Although this example concerns the lookup of a DNS-style name, the sequence and execution of operations is nearly identical for a GDS name or a hierarchical cell. If the GDA recognizes that a name is a GDS-style name, it passes the name to a GDS server rather than to a DNS server.
  6. DNS looks up and returns to the GDA information that is associated with the **widget.com** cell entry. The information includes the addresses of servers that maintain replicas of the root directory of the /**.../widget.com** cell namespace.
  7. The GDA passes the information about the foreign cell to the clerk.
  8. The clerk contacts the CDS server on Node E in the foreign cell, passing it a lookup request.
  9. The Node E server's clearinghouse contains a replica of the root directory; the server looks up the entry for **printsrv1** in the root and passes the requested information to the clerk on Node A. For

simplicity, this example shows the clerk contacting only one server in the foreign cell. While resolving a full name, the clerk may actually receive referrals to several servers in the foreign cell.

10. The clerk passes the information to the client application that requested it.

Notes:

- Both of the previous examples (Figure 5-8 on page 5-35 and Figure 5-9 on page 5-36) represent initial lookups. The CDS clerk caches the locations of GDAs when it discovers them. The clerk also caches the addresses of servers in foreign cells that it learns about enabling it to contact the foreign servers directly on subsequent requests for names in the same cell.
- A GDA knows its own cell name and can therefore avoid contacting a global directory service to look up names in its own cell. Furthermore, the GDA can also recognize whether a cell name conforms to GDS or DNS naming syntax and it uses that knowledge to route a lookup request to the appropriate global directory service.

Keeping Data Consistent Through Timestamps

CDS uses several timestamps to help ensure the consistency and accuracy of data. The following two timestamps exist for every entry:

- Creation Timestamp (CTS)
- Update Timestamp (UTS)

CDS assigns a CTS to everything that is in a cell namespace — clearinghouses, directories, object entries, soft links, and child pointers. The CTS is a unique value reflecting the date, time, and location where a clearinghouse, directory, or entry in a directory was created. It consists of two parts — a time portion and the system identifier of the node on which the name was created. The two parts guarantee uniqueness among timestamps that are generated on different nodes.

During propagation of a new or changed name to each replica of the directory where it was created, every CDS server checks the validity of the CTS before accepting the new name.

The UTS reflects the most recent change that was made to any of the attributes of a clearinghouse, directory, object entry, soft link, or child pointer. When a CDS server receives an update to an existing entry in a directory, it checks the validity of the UTS before accepting the update.

Directories and replicas have several other timestamps that CDS uses when determining whether to skulk a directory or make a change in a directory. See the **directory show** reference pages in the *DCE for AIX Administration Command Reference* for information about other timestamp attributes used by CDS.

Listing the Contents of Directories

Introduction — Listing Directory Contents

The **dcecp** program provides a **directory list** command that allows you to display a list of the descendants of a directory within the cell namespace. A directory's descendants are all the child pointers, clearinghouses, object entries, and soft links existing in it.

More Detailed Information — Listing Directory Contents

By default, the **directory list** command displays the full names of the objects (the object names preceded by /**...**/*pathname*) contained in the directory. To list only the RDNs of the objects, enter the **directory list** command with the **–simplename** option.

To display the names of a particular kind of directory descendant only, you include the appropriate option with the **directory list** command.

Displaying the Attribute Values of CDS Names

To display any or all of the current values of the attributes associated with the names in a namespace (except for clerks or servers), use the **dcecp** program's **show** command.

The basic syntax of the **show** command is as follows:

object-type **show** *object-name*

where *object-type* is the type of CDS object about which you want to display information, and *object-name* is a complete directory specification terminating with a simple name (that is, the full CDS name) of the object you are inquiring about.

Displaying Clerk and Server Attribute Information

To show the values of the attributes associated with clerk and server entries in the cell namespace, you must use the **cdscp** program. The basic syntax of this program's **show** command is as follows:

show *entity-type entity-name*

where *entity-type* is the type of CDS object about which you want to display information, and *entity-name* is a complete directory specification terminating with a simple name (that is, the full CDS name) of the object about which you are inquiring.

Permissions Required — Listing Directory Contents

- To use the **directory list** command, you must have read permission to the CDS names that you want to display.
- To use the **show** command, you must have read permission to the CDS name that you want to display.

Restrictions — Listing Directory Contents

You are not permitted to use wildcard characters in the simple names of clerks and servers on the **show** command line.

Examples — Listing Directory Contents

- For a complete listing of a directory's contents, enter the **directory list** command with the name of the directory or directories whose contents you want to view, as in the following example:

```
dcecp> directory list /.:/eng
/.../eng_cell.osf.org/hosts/eng/aud-acl /.../eng_cell.osf.org/hosts/eng/aud-svc\
/.../eng_cell.osf.org/hosts/eng/cds-clerk /.../eng_cell.osf.org/hosts/eng/cds-server\
 /.../eng_cell.osf.org/hosts/eng/dts-entity /.../eng_cell.osf.org/hosts/eng/\
profile /.../eng_cell.osf.org/hosts/eng/self /.../eng_cell.osf.org/hosts/\
eng/CDS_CTS /.../eng_cell.osf.org/hosts/eng/CDS_UTS
dcecp>
```

- Enter the following command to display the names of all the soft links that are stored in the /**.:/eng** directory:

```
dcecp> directory list  /.:/eng/ -links
/.../eng_cell.osf.org/hosts/eng/CDS_CTS /.../eng_cell.osf.org/hosts/\
eng/CDS_UTS
```

- In the following example, the **show** command is used to display the current values of the **CDS_CHDirectories** attribute associated with the /**.:/Chicago2_CH** clearinghouse. The display returned by the command shows two values for the attribute, each value having two parts. The parts of the attribute value are UUID of Directory and Name of Directory. The **show** command displays the values separately. For each value, it first lists the attribute name on a line ending with a colon, then the parts of the value.

```
dcecp> clearinghouse show  /.:/Chicago2_CH
{RPC_ClassVersion
 {01 00}}
{CDS_CTS 1994-01-24-07:12:51.966-05:00I0.000/00-00-c0-f7-de-56}
{CDS_UTS 1994-02-03-07:17:35.794-05:00I0.000/00-00-c0-f7-de-56}
{CDS_ObjectUUID 0094e40e-bb43-1d43-9e0a-0000c0f7de56}
{CDS_AllUpTo 1994-02-03-09:17:06.393-05:00I0.000/00-00-c0-f7-de-56}
{CDS_DirectoryVersion 3.0}
{CDS_CHName /.../Chicago2/Chicago2_CH}
{CDS_CHLastAddress
 {Tower ncacn_ip_tcp:130.105.5.16[]}}
{CDS_CHLastAddress
 {Tower ncadg_ip_udp:130.105.5.16[]}}
{CDS_CHState on}
{CDS_CHDirectories
 {dir_uuid 00595ca5-bb46-1d43-9e0a-0000c0f7de56}
 {directory /.../Chicago2}}
{CDS_CHDirectories
 {dir_uuid 00888574-bb53-1d43-9e0a-0000c0f7de56}
 {directory /.../Chicago2/subsys}}
{CDS_CHDirectories
 {dir_uuid 0069ff14-bb55-1d43-9e0a-0000c0f7de56}
 {directory /.../Chicago2/subsys/dce}}
{CDS_CHDirectories
 {dir_uuid 0023cc38-bb56-1d43-9e0a-0000c0f7de56}
 {directory /.../Chicago2/subsys/dce/sec}}
{CDS_CHDirectories
 {dir_uuid 0026d57c-bb57-1d43-9e0a-0000c0f7de56}
 {directory /.../Chicago2/hosts}}
{CDS_ReplicaVersion 3.0}
{CDS_NSCellname /.../Chicago2}
dcecp>
```

- In the following example, the **show** command displays all of the object entries that are stored in the /**.:/sales** directory:

```
dcecp> object show  /.:/sales
{CDS_CTS 1994-06-23-15:56:44.734+00:00I0.000/08-00-2b-0f-59-bf}
{CDS_UTS 1994-08-08-22:23:54.226+00:00I0.000/08-00-2b-0f-59-bf}
{CDS_ClassVersion 1.0}
dcecp>
```

- The following command displays all of the soft links stored in the /**.:/mfg** directory:

```
dcecp> link show /.:/mfg
{CDS_CTS 1994-06-23-15:56:44.734+00:00I0.000/08-00-2b-0f-59-bf}
{CDS_UTS 1994-08-08-22:23:54.226+00:00I0.000/08-00-2b-0f-59-bf}
{CDS_LinkTarget = /.../abc/mfg/robotics_controller1}
dcecp>
```

- In the following example, the **show** command displays the current values of all attributes that are associated with the local clerk:

```
cdscp> show clerk

                    SHOW
                   CLERK
                      AT   1991-10-15-15:56:50
           Creation Time = 1991-10-09-17:03:32.32
 Authentication failures = 0
         Read Operations = 1068
              Cache Hits = 137
           Cache bypasses = 433
        Write operations = 1250
Miscellaneous operations = 590
```

Locating Names

Introduction — Locating Names

As Figure 5-17 on page 5-66, Figure 5-18 on page 5-67, Figure 5-19 on page 5-68, and Figure 5-20 on page 5-69 illustrate, CDS finds information about the physical location of a resource by looking up one or more attributes that are associated with its name. First, the clerk must know how to find the name. If a name does not exist in the clerk's cache, the clerk must be able to identify at least one CDS server to contact in search of the name.

More Detailed Information — Locating Names

The clerk can learn about CDS servers and their locations in any of three ways:

- **Through the Solicitation and Advertisement Protocol** — Clerks and servers on the same Local Area Network (LAN) communicate by using the solicitation and advertisement protocol. A server broadcasts messages at regular intervals to advertise its existence to clerks on its LAN. The advertisement message contains information about the cell that the server belongs to, the server's network address, and the clearinghouse it manages. Clerks learn about servers by listening for these advertisements on the LAN. A clerk also sends out solicitation messages that request advertisements at startup.
- **During a Lookup** — If a clearinghouse does not contain a name for which the clerk is searching during a lookup, the server managing that clearinghouse gives the clerk as much data as it can about where to search for the name. If a clearinghouse contains replicas that are part of the full name being looked up, but not the replica containing the target simple name, it returns data from a relevant child pointer in the replica it does have. The data helps the clerk find the next child directory in the path toward the target simple name. The child pointer's **CDS_Replicas** attribute contains this data, in the form of clearinghouse names and binding information.
- **The cdscache create Command** — A DCE administrator can run the **dcecp** program's **cdscache create** command to create knowledge in the clerk's cache about a server. This command is useful when the server and clerk are separated by a Wide Area Network (WAN) and the clerk cannot, therefore, learn about the server from advertisements on a LAN.

Figure 5-10 on page 5-41 provides an example of how the clerk works downward from the root of the cell namespace to locate an object entry. The object entry, /**.:/Sales/Spell**, describes a spell-checking server at a company's London sales headquarters.

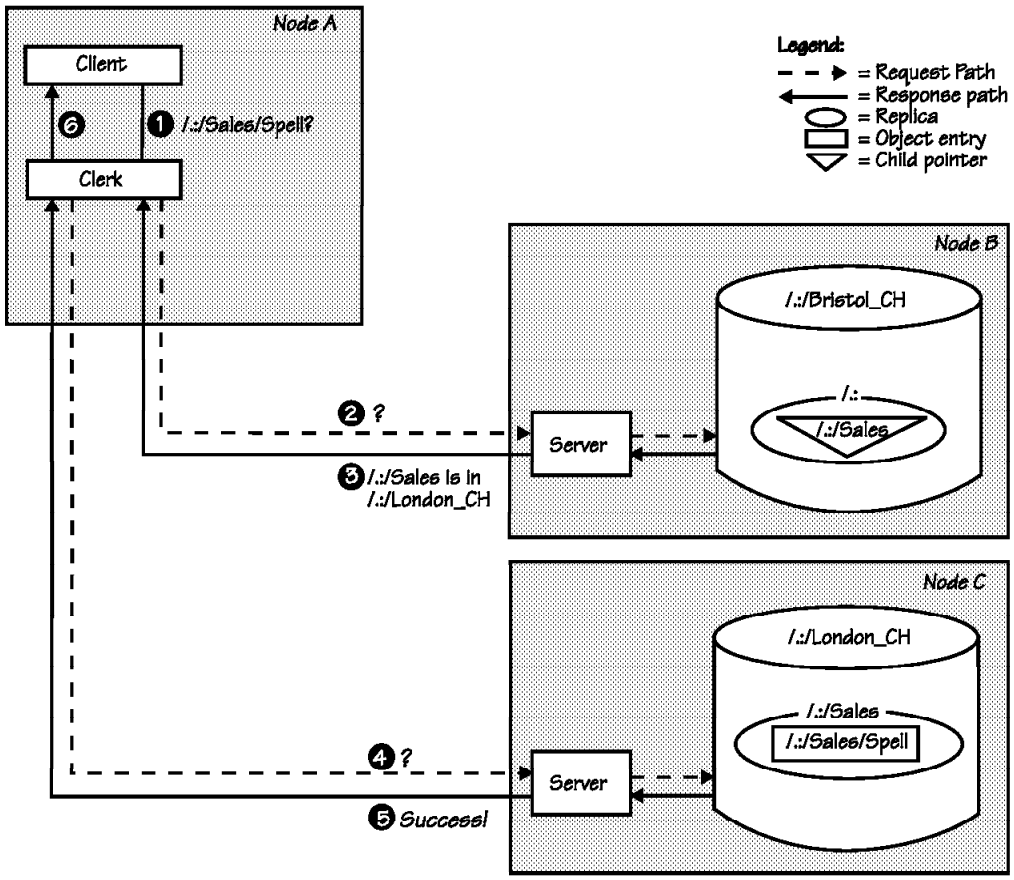


Figure 5-10. How the Clerk Finds a Name

As shown in Figure 5-10, the clerk locates the desired object entry by performing the following steps:

1. On Node A, a spell-checking application requests the network address of the /**.:/Sales/Spell** server. The clerk does not have that name in its cache and the only clearinghouse it knows about so far is the /**.:/Bristol_CH** clearinghouse on Node B.
2. The clerk contacts the server on Node B with the lookup request.
3. The /**.:/Bristol_CH** clearinghouse does not contain the target object entry, but it does contain a replica of the root directory. From the /**.:/Sales** child pointer in the root, the clerk can learn how to contact clearinghouses that have a replica of the /**.:/Sales** directory. The server on Node B returns this data to the clerk, informing it that a replica of /**.:/Sales** is in the /**.:/London_CH** clearinghouse on Node C.
4. The clerk contacts the server on Node C with the lookup request.
5. The /**.:/Sales** replica in the clearinghouse on Node C contains the /**.:/Sales/Spell** object entry, so the server passes the address of the spell-checking server to the clerk.
6. The clerk returns the information to the client application, which can now make a remote call to the spell-checking server.

Long lookups, as illustrated in Figure 5-10, do not normally happen often after a clerk establishes its cache and becomes more knowledgeable about clearinghouses and their contents. However, the figure illustrates the resources and connections that could be involved in an initial lookup. The figure also illustrates the importance of maintaining connectivity between parent and child directories in the namespace. If the directory path is broken or a clearinghouse is unreachable, a clerk may not be able to find a name.

Managing CDS with dcecp Commands

Introduction — Managing CDS with dcecp Commands

"**dcecp** Commands Usage" on page 2-19 describes the **dcecp** program and its command syntax, so this section does not repeat this information. It simply describes some **dcecp** commands available for managing CDS.

More Detailed Information — Managing CDS with dcecp Commands

CDS Managed Objects

The following describes the commands that the **dcecp** program supplies specifically for managing CDS. The **dcecp** program commands operate on the following objects representing CDS entities:

directory This object represents a CDS directory. The directory can be a parent or child directory, or a master or read-only replica of the parent or child directory. In addition to child directories, a CDS directory can contain soft links and object entries for other CDS resources.

link This object represents a soft link in a CDS directory. A soft link is a pointer to (alternate name for) a child directory, object entry, or other soft link.

object This object represents a object entry, which is the name of a CDS resource in the cell namespace. Some object entries name resources that CDS clients can access (for example, a disk, machine, or application). Others name resources solely for internal use by CDS (for example, servers and clearinghouses).

clearinghouse This object represents a CDS clearinghouse. A clearinghouse is a database that is located on a CDS server machine for use by servers.

cdscache This object represents a CDS cache. A CDS cache is a collection of information about servers, clearinghouses, and other CDS resources that a CDS clerk establishes on the local system for its reference.

CDS Command Operations

The **dcecp** program's commands for managing CDS perform the operations shown in the following list.

Command Operations	Definitions
add	Adds a child directory to a parent in the cell namespace.
catalog	Displays a list of a DCE cell's alias names.
create	Creates an object in the cell namespace. The object type can be a directory, object entry, soft link, clearinghouse, or CDS cell alias.
delete	Deletes an object in the cell namespace. The object type can be a directory, object entry, soft link, clearinghouse, or CDS cell alias.
disable	Removes the knowledge of a clearinghouse from the server running on the local machine.
dump	Displays an in-core dump of a CDS cache.
help	Displays a help message for a CDS object type, describing the operations that it performs or operations that can be performed on it. The object type can be a directory, object entry, soft link, clearinghouse, or CDS cache.

list	Displays the names of all CDS objects contained in a directory.
merge	Copies the contents of a directory into another directory.
modify	Modifies the attribute information for a CDS object type. The object type can be a directory, object entry, or soft link.
operations	Displays the operations that a CDS object type can perform or can have performed on it. The object type can be a directory, object entry, soft link, or clearinghouse.
ping	Checks if all or selected servers are running in a DCE cell.
remove	Removes a child directory from a parent in the cell namespace.
set	Sets the primary name of a cell to a different alias or creates a hierarchical relationship between two cells through an alias.
show	Displays the attribute information for a CDS object type. The object type can be a directory, object entry, soft link, or clearinghouse.
synchronize	Tells a child or parent directory to synchronize with its replicas (perform a skulk).

CDS Object Attributes

Every CDS object has attributes, which are pieces or sets of data associated with the object. Attributes can reflect or affect the operational behavior of the object. Some attributes are created and modified only by CDS; you can modify others as needed for your environment. For a complete list of the attributes of a particular CDS object, refer to its reference page. Also, you can use the **dcecp** program's **show** operation for most objects to display the names and values of all attributes or specific attributes of the objects.

Managing Soft Links

Introduction — Managing Soft Links

A soft link is an alternate name or alias with which you can refer to another existing name in a namespace. Soft links allow users and client applications to refer to a particular directory, object entry, or soft link by more than one name.

In general, you should create soft links to assign alternate names to particular network resources or to make minor changes to the original names of directories in your namespace hierarchy. You should avoid using soft links to completely redesign your namespace.

More Detailed Information — Managing Soft Links

Creating a Soft Link

Use the **dcecp** program's **link create** command to create a soft link. In addition to the name for the new soft link, you must specify the soft link's destination name, or the existing name to which the new soft link points, with the **–to** option. You can specify any name in the local cell namespace or in any foreign cell namespace (including another soft link) as the destination name.

Note: If you create a soft link that points to another soft link, make sure you do not create a soft link loop. A soft link loop occurs when you specify a destination name that eventually points back to the new soft link's own link name. The clerk detects this error.

All soft links that you create with the **link create** command are permanent and will never expire unless you use the command's **–timeout** option. You can use this argument to specify an expiration date and

time value in the **CDS_LinkTimeout** attribute of the soft link. Enter the expiration date and time value in the format *yyyy-mm-dd-hh:mm:ss*. For example, **CDS_LinkTimeout=(1995-08-25-16:00:00)** indicates that, if the soft link still exists (that is, has not been deleted manually) on August 25, 1995, at 4:00 p.m., CDS will automatically delete it the next time the directory in which it is stored is skulked.

If you use the **–timeout** option to specify an expiration value, you can also specify an extension value. An extension value is a period of time to be added to the expiration date and time already assigned. Enter the extension value in the format *ddd-hh:mm:ss*. For example, a value of **030-00:00:00** indicates that, if the destination name of the soft link still exists when the assigned expiration date and time are reached, CDS allows another 30 days to pass before it checks again, during a skulk, for the existence of the destination name. If, at that time, the destination name cannot be found, CDS deletes the soft link.

Changing the Destination Name of a Soft Link

Use the **dcecp** program's **link modify** command to specify a new value for the **CDS_LinkTarget** attribute and to redirect the soft link from its current destination name to some other name in the namespace.

Changing a Soft Link's Expiration or Extension Value

Use the **dcecp** program's **link modify** command to specify a new value for the expiration and extension values that are stored in a soft link's **CDS_LinkTimeout** attribute. Even if you want to modify only one of the values, you must specify both expiration and extension values in your command. You specify a new value in the same format that you used to establish the original value. Specify an expiration value in the format *yyyy-mm-dd-hh:mm:ss* and an extension value in the format *ddd-hh:mm:ss*.

Deleting a Soft Link

If you find that a permanent soft link has outlasted its original purpose or if you prefer not to wait until the assigned expiration and extension times have been reached, you can delete the soft link from the namespace yourself.

Permissions Required — Managing Soft Links

- To create a soft link, you must have insert permission to the directory in which you intend to create the soft link.
- To change a soft link's destination name, you must have write permission to the soft link.
- To change a soft link's expiration or extension value, you must have write permission to the soft link.

Examples — Managing Soft Links

- The following command creates a permanent soft link named /**.:/sales/asia** that points to a directory named /**.:/sales/eur**:

  ```
  dcecp> link create /.:/sales/asia -to /.:/sales/eur
  ```

- The following command creates a soft link named /**.:/mfg/robo1** that points to an object entry named /**.:/mfg/robotics_controller01** and sets its expiration date and time:

  ```
  dcecp> link create /.:/mfg/robo1 -to
                    /:/mfg/robotics_controller01 -timeout 1995-12-12-09:00:00
  ```

  In the preceding command, the expiration date and time placed in the **CDS_LinkTimeout** attribute value indicates that CDS will delete the soft link /**.:/mfg/robo1** on the next skulk after December 12, 1995, at 9:00 a.m.

- The following command creates a soft link named /**.:/admin/linka** that points to an object entry named /**.:/sales/discount_stats**:

```
dcecp> link create /.:/admin/linka -to /.:/sales/discount_stats
                  -timeout {1995-01-11-12:00:00 090-00:00:00}
```

In the preceding command, the expiration time placed in the **CDS_LinkTimeout** attribute value indicates that CDS will verify that the destination name /**.:/sales/discount_stats** still exists on the next skulk after January 11, 1995, at 12:00 p.m. If the destination name does not exist, CDS deletes the soft link. If the destination name still exists, the soft link remains in effect for another 90 days, as specified by the extension time specified for the **CDS_linkTimeout** attribute value **090-00:00:00**. When the 90-day extension period expires, CDS repeats the check at 90-day intervals until the destination name is deleted.

- The following command redirects a soft link named /**.:/admin/work_disk** from its current destination name to a new destination name /**.:/admin/work_disk03**:

  ```
  dcecp> link modify /.:/admin/work_disk -change {CDS_LinkTarget /.:/admin/work_disk03}
  ```

- The following command sets the expiration value of a soft link named /**.:/eng/link01** to December 31, 1995, at 12:00 p.m. In this example, no extension is currently assigned to the soft link.

  ```
  dcecp> link modify /.:/eng/link01 -change
                  {CDS_LinkTimeout 1995-12-31-12:00:0 0000-00:00:00}
  ```

- The following command changes the expiration value of a soft link named /**.:/eng/link01** to December 31, 1995, at 12:00 p.m. and sets the soft link's extension value to 90 days:

  ```
  dcecp> link modify /.:/eng/link01 -change
                  {CDS_LinkTimeout 1995-12-31-12:00:00 090-00:00:00}
  ```

- Use the **link delete** command to delete the soft link of the name that you specify. To delete a soft link, you must have delete permission to the soft link or administer permission to the directory that stores the soft link. The following command deletes a soft link named /**.:/dist/pointer_1**:

  ```
  dcecp> delete link /.:/dist/pointer_1
  ```

Managing the Global Directory Agent

Introduction — Managing the GDA

Configure the GDA using the DCE configuration program; the GDA requires little management after it is configured. See *DSS for AIX Up and Running!* for details on configuring the GDA.

The GDA is typically started and stopped automatically by scripts that execute as part of normal system startup and shutdown procedures. Sometimes, however, you may want to use commands to stop and restart a GDA. After you have configured GDA with the DCE configuration program, you can use these steps to start and stop GDA. The GDA runs as a process called **gdad**.

Steps — Managing the GDA

- To start the **gdad** process, follow these steps:

  1. Make sure that a CDS server is already running somewhere within the cell.
  2. Log into the system as superuser (**root**).
  3. Enter the following command to see if the **dced** process is already running:

     ```
     # ps
     ```

     If the **dced** process displays as an active process, proceed to Step 4 on page 5-46. If the **dced** process does not appear on the list of active processes, enter the following command to start the process:

```
# dced
```

4. Enter the following command to start the **cdsadv** process:

   ```
   # cdsadv
   ```

5. Enter the following command to start the **gdad** process:

   ```
   # gdad
   ```

- To stop the GDA, enter the following command:

  ```
  # kill pid
  ```

  where *pid* is the process identifier of the **gdad** process.

Merging CDS Directories

Introduction — Merging CDS Directories

To merge two directories, you must first perform a skulk operation and then use the **directory merge** command.

More Detailed Information — Merging CDS Directories

To preserve the access by principals to the merged information in the target directories, the ACLs on the newly created objects at the target location need to match those of the objects in the source directories. Because the **directory merge** command does not recreate the source ACLs on the CDS objects at the new location, you may need to modify the target ACLs after the merge operation. To modify these ACLs, use the **dcecp** program's **acl replace** or **acl modify** command, depending on whether you want to replace an entire ACL or just modify ACL entries.

Steps — Merging CDS Directories

The following procedure merges the source directory /**.:/eng** into the target directory /**.:/rnd**:

1. Perform a skulk on the /**.:/eng** directory before merging it with the /**.:/rnd** directory. This synchronization of the source directory's replicas can prevent errors that cause the merge operation to fail.

   ```
   dcecp> directory synchronize  /.:/eng
   ```

2. Run the **directory merge** command to merge the /**.:/eng** and /**.:/rnd** directories:

   ```
   dcecp> directory merge  /.:/eng  -into  /.:/rnd
   ```

   Note that the **directory merge** command merges only the immediate contents of the source directory named in the command line argument (that is, the object entries, soft links, and child directories in these directories). To copy the descendants of any child directories of a directory to a target location, you must use the **–tree** option of the command. For example, if the /**.:/eng** directory in the previous example included the child directories **dev** and **qa** and you wanted to merge the contents of these directories into the target directory /**.:/rnd**, you would enter the command line:

   ```
   dcecp> directory merge /.:/eng -into /.:/rnd -tree
   ```

   By default, the **directory merge** command places all object entries, soft links, and child directories in the target directory's master clearinghouse. You can, however, place child directories in another clearinghouse. To do this, you use the **–clearinghouse** option of the command to specify the name of the other clearinghouse.

Note that you are allowed to specify only one alternate clearinghouse in the **–clearinghouse** option. To place child directories in different alternate clearinghouses, you must issue separate **directory merge** commands for each clearinghouse, or you must issue a single **directory merge** command to place all the child directories in one clearinghouse, then relocate the directories after the merge operation.

Note: The CDS objects created by the **directory merge** command retain all of the writable attribute values and some of the read-only attribute values of the source objects. However, these objects do not inherit the ACLs of the source objects. If the merged object is a directory, the **directory merge** command gives it the default ACLs of the initial container. If the merged object is any other CDS object type, the **directory merge** command gives it the default ACLs of the initial object.

If the **directory merge** command encounters problems with the merge operation, it behaves in one of two ways. If you include the **–nocheck** option, the command does not check for errors before performing the operation. It proceeds immediately to perform the operation, and, if it encounters an error, stops. If you omit the **–nocheck** option, the command checks for certain error conditions before starting the merge. If it finds errors, it displays messages for the errors and stops, otherwise it proceeds with the merge.

Error messages returned by the **directory merge** command identify the CDS entity causing the problem and provide a brief description of the problem. You should fix any problems that the command encounters, before running it again. See “Error Handling” on page 5-33 for more information on the types of errors that can occur during a merge operation.

3. After the merge operation, the /**.:/eng** directory (and its contents) still exists at the source location. Run the following commands to delete the /**.:/eng** directory from its original location and create a soft link named /**.:/eng** in place of the deleted directory. The soft link will redirect lookups of the **obj1** and **link1** object entries to their new locations in the /**.:/rnd** directory.

   It is recommended that you perform a skulk on a source directory before deleting it. This synchronization of the directory's replicas can prevent errors that cause the delete operation to fail.

   The sequence of commands to synchronize and delete the /**.:/eng** directory and then create soft links for the former contents are as follows:

   ```
   dcecp> directory synchronize  /.:/eng
   dcecp> directory delete  /.:/eng  -tree
   dcecp> link create  /.:/eng  -to  /.:/rnd
   ```

   The **directory delete** command invoked with the **–tree** option deletes a directory and all the object entries, soft links, and child directories beneath that directory. If you use the **directory delete** command without the **–tree** option, all of the directories to be deleted must be empty, or errors will occur.

Appending CDS Directories

The following procedure appends the source directory /**.:/eng** to the /**.:/rnd** directory (that is, copies the /**.:/eng** directory into the empty target directory /**eng** under the /**.:/rnd** directory):

1. Run the **directory create** command to create a new empty directory named /**.:/rnd/eng** into which the contents of the source directory /**.:/eng** can be placed:

   ```
   dcecp> directory create  /.:/rnd/eng
   ```

   By default, the **directory create** command creates new directories in the same clearinghouse as the parent directory. To create a directory in another clearinghouse, you must use the **–clearinghouse** option of the command to specify the other clearinghouse.

2. Perform a skulk on the /**.:/eng** directory before appending it to the /**.:/rnd** directory. This synchronization of the source directory's replicas can prevent errors that cause the append operation to fail:

   ```
   dcecp> directory synchronize  /.:/eng
   ```

3. Run the **directory merge** command to append the source directory /**.:/eng** to the /**.:/rnd** directory (or merge it into the new /**.:/rnd/eng** directory):

   ```
   dcecp> directory merge  /.:/eng  -into  /.:/rnd/eng
   ```

   If the source directory contains a child directory whose contents you want to copy over, you must specify the **–tree** option in the **directory merge** command line. Additionally, you need to specify the **–clearinghouse** option if you want to place the child directory and its contents in a different clearinghouse from the /**.:/rnd/eng** directory.

   If the merge operation is not successful, you can delete any partially merged information at the target location and run the command again. Be sure, though, to delete any duplicate names and to make certain that connectivity to the affected clearinghouses can be maintained.

   Note: The CDS objects created by the **directory merge** command retain all of the writable attribute values and some of the read-only attribute values of the source objects. However, these objects do not inherit the ACLs of the source objects. The ACLs on the target objects are either those that are inherited from the initial container (the parent directory into which the objects are merged) or the initial object.

4. After the append operation, the /**.:/eng** directory (and its contents) still exists at the source location. You need to delete the /**.:/eng** directory from its original location and create a soft link named /**.:/eng** in place of the deleted directory. The soft link will redirect lookups of the **obj1** and **link1** object entries to their new locations in the /**.:/rnd/eng** directory.

   It is recommended that you perform a skulk on a source directory before deleting it. This synchronization of the directory's replicas can prevent errors that cause the delete operation to fail.

   The sequence of **dcecp** program commands for removing the /**.:/eng** directory from the source location is the following:

   ```
   dcecp> directory synchronize  /.:/eng
   dcecp> directory delete  /.:/eng
   dcecp> link create  /.:/eng  -to  /.:/rnd/eng
   ```

Examples — Merging CDS Directories

- Figure 5-11 shows the structure of the example namespace before and after the merge operation in our example.

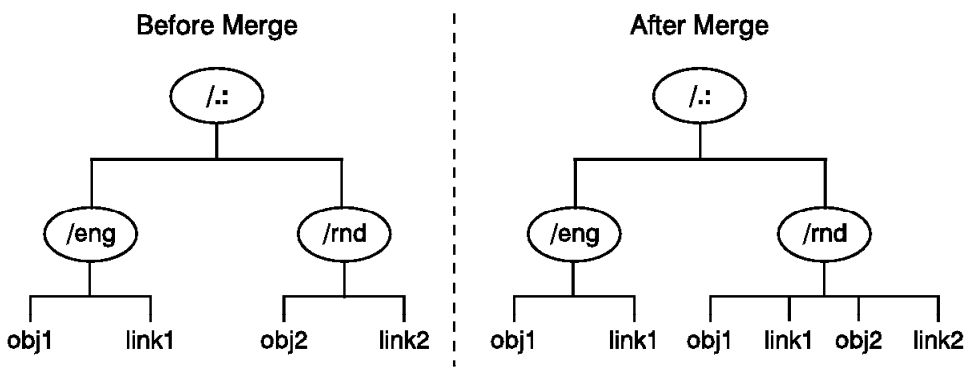


Figure 5-11. Example Namespace Before and After the Merge Operation

- Figure 5-12 on page 5-49 shows the structure of our example namespace before and after the append operation.

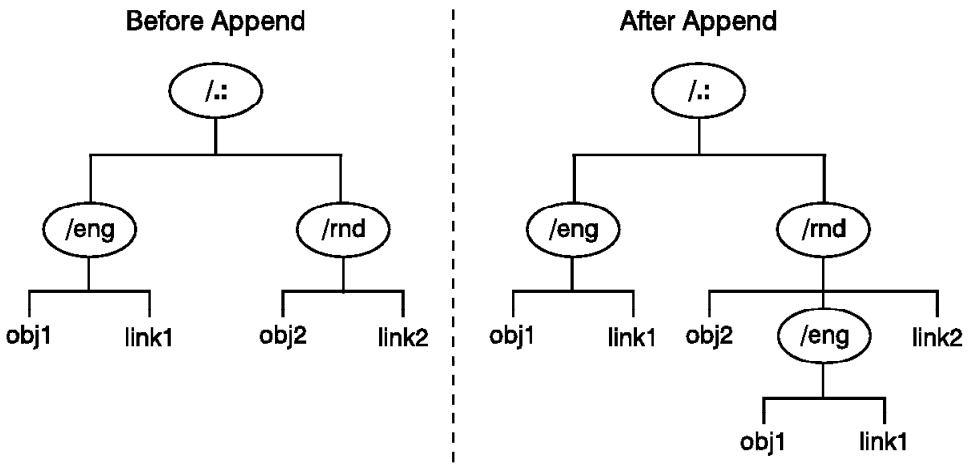


Figure 5-12. Example Namespace Before and After the Append Operation

Merging CDS Directories into a Foreign Cell

Introduction — Merging Directories into Foreign Cells

You can use the **directory merge** command to merge CDS directories into the namespace of a foreign cell. In general, the procedure you follow is the same as the procedure you use to merge directories or subtrees in the same namespace. There are, however, some additional considerations to keep in mind:

- You need to establish cross-cell authentication in advance.
- You need to merge the entire directory hierarchy in the source and target cells.

Also, you need to modify the ACLs of the newly created target objects as when you merge directories in the same namespace.

More Detailed Information — Merging Directories into Foreign Cells

If you want users and applications in the source cell to be able to continue accessing their merged information in the target cell conveniently, make sure that an agreement of cross-cell authentication exists between the source cell and foreign (target) cell. Otherwise, principals from the source cell requesting newly merged information will not be permitted to communicate with the target cell. See the “Creating Trust Relationships” on page 7-60 on how to set up cross-cell authentication.

Steps — Merging Directories into Foreign Cells

To merge CDS data into the namespace of a foreign cell, follow these steps:

1. While logged into a privileged account (**cell_admin** or a member of **cds-admin** group) on the target machine in the foreign cell, run the **directory merge** command to merge the contents of the source cell's directory with an existing directory.
2. If you intend to continue accessing the merged information from the source cell, delete the uppermost directory in the source subtree and replace the deleted information with a single soft link of the same name as that directory. This redirects lookups of the information to its new location in the foreign cell.

Modifying a Directory's Convergence

Introduction — Modifying a Directory's Convergence

The value assigned to a directory's **CDS_Convergence** attribute determines how frequently the server that stores the master replica of the directory initiates a skulk of the directory's replica set. A directory's convergence can be set to a value of **high**, **medium**, or **low**.

More Detailed Information — Modifying a Directory's Convergence

A directory that is set to a convergence value of **high** is skulked at least once every 12 hours. If an update is made to the directory, the server that stores the master replica immediately attempts to propagate the new information to the entire replica set. If this update propagation fails, the server schedules a skulk of the directory to begin within the hour. If this initial skulk fails, additional skulks are initiated at 1-hour intervals until the skulk succeeds.

A directory set to a convergence value of **medium** is skulked at least once every 12 hours. If an update is made to the directory, the server that stores the master replica immediately attempts to propagate the new information to the entire replica set. If the propagation fails, the server waits for the next skulk to synchronize the replica set.

A directory that is set to a convergence value of **low** is skulked at least once every 24 hours. The server on which a low-convergence directory resides makes no immediate attempt to propagate updates and waits for the next skulk to synchronize the replica set.

Every newly created directory inherits the convergence value of its parent directory. When you create a namespace, the root directory is automatically assigned a convergence value of **medium**. Unless you change this value, or the convergence values of any lower-level directories after you create them, all directories that you create under the root also have a convergence value of **medium**. For most directories, you never need to modify this value. However, you may occasionally find it useful to set a directory's convergence to **high** or **low**.

Before You Modify a Directory's Convergence

Before you modify a directory's convergence, verify the current convergence value of the directory. To do this, use the **directory show** command to display the directory's attribute values and look at the **CDS_Convergence** attribute value.

Entering the directory modify Command

Use the **directory modify** command with the **–change** option to assign a value of **high**, **medium**, or **low** to a directory's **CDS_Convergence**. For example, the following command sets the convergence value of the /**.:/sales/us** directory to **high**:

```
dcecp> directory modify /.:/sales/us -change {CDS_Convergence high}
```

Permissions Required — Modifying a Directory's Convergence

To modify a directory's convergence, you must have write permission to the directory.

Modifying a Directory's Replica Set

Introduction — Modifying a Directory's Replica Set

A directory's replica set always contains a master replica. It can also contain other read-only replicas. The values that are stored in the **CDS_Replicas** attribute that is associated with a directory contain information that describes the directory's replica set, including how many replicas exist, their replica types, and the name of the clearinghouse where each of the replicas is stored. You can use the **cdscp** program's **set directory to new epoch** command to overwrite the current values that are stored in the directory's **CDS_Replicas** attribute and to perform either or both of the following tasks in a single command:

- Designate a new master replica in a directory's replica set.
- Exclude a replica from a directory's replica set.

Note: As part of the **set directory to new epoch** command, CDS initiates an immediate skulk on the directory to distribute modifications to all members of the replica set as soon as possible.

More Detailed Information — Modifying a Directory's Replica Set

Before You Modify a Replica Set

Before you modify a directory's replica set, you need to know how many replicas exist, their replica types, and the name of the clearinghouse where each of the replicas is stored. The command that you use to modify a directory's replica set does not allow you to accidentally leave a replica out of the new set. You must explicitly list all existing replicas that are in the set. You can include or exclude any replica from the new set, but you must account for all replicas. Only one of the replicas that you include in the new set can be designated as the master replica.

To display the names of all directory's replicas, use the **dcecp** program's **directory show** command. This command queries the directory's **CDS_Replicas** attribute to gather this information. See “Displaying the Structure and Contents of a Namespace” on page 1-30, for complete information on how to use the **dcecp** program's **directory show** command.

Designating a New Master Replica

Sometimes, for configuration management reasons, you may want to designate a different replica as a directory's master replica. For example, you can specify a new master replica when:

- A server system clearinghouse that contains one or more master replicas will be out of service for an extended period of time or will be removed permanently from the network;
- A clearinghouse that stores one or more master replicas will be deleted from the namespace; or
- You want to locate a master replica closer to the source originating the majority of directory updates.

Use the **cdscp** program's **set directory to new epoch** command to designate a new master replica for the directory that you specify.

Excluding a Replica from a Replica Set

When the clearinghouse where the replica is stored unexpectedly becomes unavailable, you can temporarily exclude a replica from its replica set. This makes it possible for CDS to complete skulks of the directory during the time the excluded replica is unavailable.

To exclude a replica from a replica set, use the **cdscp** program's **set directory to new epoch** command with the **exclude** argument to rebuild a directory's replica set (excluding the replica that you specify). Remember that you must account for all existing replicas in the command.

Note: Always reintroduce excluded replicas to their replica sets as soon as possible after the clearinghouse in which they reside again becomes available.

Permissions Required — Modifying a Directory's Replica Set

To modify a directory's replica set, you must have the following permissions:

- You must have administer permissions to the directory. The server principal also needs administer, read, and write permission to the directory.
- When designating a new master replica, you also need write permission to the clearinghouse that stores the current master replica. The server principal needs write permission to the clearinghouse that stores the read-only replica that you intend to designate as the new master replica.

  The server principal on the server where the new master replica will be located needs administer, read, and write permissions to the directory.

When you know which replicas to include and exclude, and have changed permissions that need to be changed, issue the **set directory to new epoch** command to modify a directory's replica set. Instructions for your two options — designating a new master replica and excluding an existing read-only replica — are given in the sections that follow.

Examples — Modifying a Directory's Replica Set

- In this example, the replica set of the /**.:/eng** directory consists of three replicas — the master replica (stored in clearinghouse /**.:/NY1_CH**), a read-only replica stored in clearinghouse /**.:/NY2_CH**, and a read-only replica stored in clearinghouse /**.:/Chicago1_CH**, as shown in Figure 5-13.

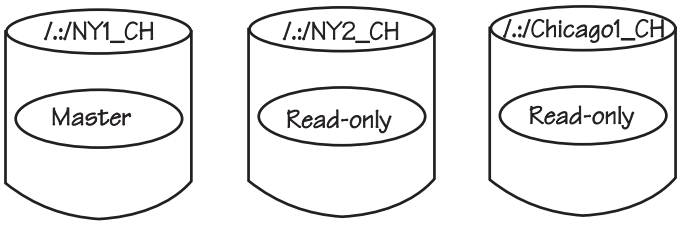


Figure 5-13. Example Replica Set

- The following command designates the read-only replica that is stored in clearinghouse /**.:/Chicago1_CH** as the directory's new master replica, designates the former master replica (stored in clearinghouse /**.:/NY1_CH**) as a read-only replica, and leaves the read-only replica stored in clearinghouse /**.:/NY2_CH** as it is.

  ```
  cdscp> set directory /.:/eng to new epoch master
                       /.:/Chicago1_CH readonly /.:/NY1_CH /.:/NY2_CH
  ```

  Figure 5-14 shows the result of the preceding command.

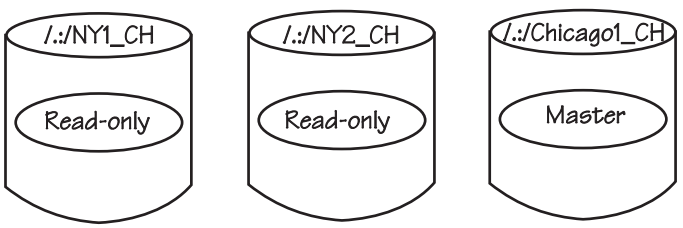


Figure 5-14. Example Replica Set After Master Redesignation

- In the following example, the replica set of the /**.:/eng** directory consists of three replicas — the master replica (stored in clearinghouse /**.:/Chicago1_CH**) and the read-only replicas (stored in clearinghouses /**.:/NY1_CH** and /**.:/NY2_CH**).

  In this case, the /**.:/NY1_CH** clearinghouse is cut off from the network because of accidental damage to the network transmission lines. Connectivity to the clearinghouse will not be restored for several days. During this period, skulks of the /**.:/eng** directory will fail unless you temporarily exclude the read-only replica that is stored in clearinghouse /**.:/NY1_CH**.

  To make it possible for skulks of the /**.:/eng** directory to succeed during the repair period, enter the following command to overwrite the current values of the /**.:/eng** directory's **CDS_Replicas** attribute with new values that include only the replicas that are stored in the /**.:/NY2_CH** and /**.:/Chicago1_CH** clearinghouses:

  ```
  cdscp> set directory /.:/eng to new epoch master /.:/Chicago1_CH readonly
                      /.:/NY2_CH exclude /.:/NY1_CH
  ```

  Figure 5-15 shows the result of the preceding command.

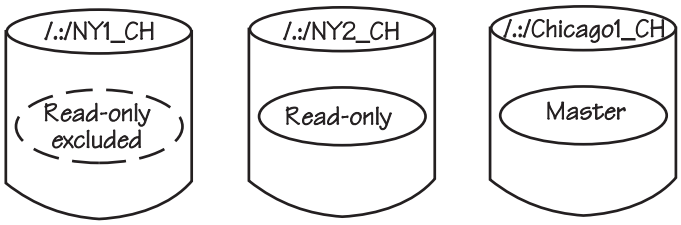


Figure 5-15. Example Replica Set After Replica Exclusion

When connectivity with the /**.:/NY1_CH** clearinghouse is re-established, enter the following command to reintroduce the read-only replica that is stored in clearinghouse /**.:/NY1_CH** to the replica set:

```
cdscp> set directory /.:/eng to new epoch master /.:/Chicago1_CH readonly
                    /.:/NY1_CH /.:/NY2_CH
```

Monitoring Clerk Communications with Specific Clearinghouses

Introduction — Monitoring Clerk Communications

Every CDS clerk maintains a separate set of clearinghouse counters to keep track of read, write, and other operations that it directs to each of the clearinghouses with which it communicates. These records collectively represent the CDS cached clearinghouse entity for a particular clerk.

You can monitor a clerk's cached clearinghouse counters so that you can look at the distribution of the clerk's transactions to each of the clearinghouses that it uses and to find out where a clerk's requests are most often directed. To do this, use the **dcecp** program's **cdscache show** command with the **–clear** option.

Examples — Monitoring Clerk Communications

- To display the cached clearinghouse counters that are maintained by the local clerk for the /**.:/NY1_CH** clearinghouse you enter:

  ```
  dcecp> cdscache show /.:/NY1_CH -clear
  ```

- The following command displays the cached clearinghouse counters for all clearinghouses used by the local clerk:

```
dcecp> cdscache show /.:/* -clear
```

Monitoring Clerk, Server, and Clearinghouse Counters

Introduction — Monitoring Counters

Every clerk, server, and clearinghouse maintains a set of attributes called *counters* to keep track of the read, write, and other operations that it performed (or that were performed on it) since it was last started up. You can monitor these counters to determine the type and volume of the CDS traffic that is being generated on your network. Clerk, server, and clearinghouse counters are fully described in the *DCE for AIX Administration Command Reference*.

More Detailed Information — Monitoring Counters

To Display Clerk Counters

Use the CDS control program's (**cdscp**) **show clerk** command to display the current counter values for a clerk.

To Display Server Counters

Use the **cdscp** program's **show server** command to display the current counter values for a server.

To Display Clearinghouse Counters

Use the **dcecp** program's **clearinghouse show** command with the **–counters** option to display the current counter values for a specified clearinghouse.

Examples — Monitoring Counters

- The following command displays the current values of all attributes that are associated with a clerk:

  ```
  cdscp> show clerk
  ```

- The following command displays the current values of all attributes that are associated with a server:

  ```
  cdscp> show server
  ```

- The following command displays the current value of all attributes that are associated with the remote clearinghouse /**.:/Paris1_CH**.

  ```
  cdscp> clearinghouse show /.:/Paris1_CH -counters
  ```

Overview of DCE Authorization for CDS

Introduction — Overview of DCE Authorization for CDS

CDS authorization allows you to control user access to the following CDS components:

- Names that are stored in the namespace, including clearinghouses, directories, object entries, soft links, and child pointers
- Execution of privileged CDS clerk and server commands

You control access to a name in the namespace by creating an Access Control List (ACL). An ACL contains individual entries that specify the permissions you grant a user (principal) to the name with which

the ACL is associated. The ACL entries that you create collectively determine which principals can use the name and what management operations they are allowed to perform on it.

More Detailed Information — Overview of DCE Authorization for CDS

CDS ACL management software, incorporated into all CDS clerks and servers, performs access checking for incoming CDS requests. When a principal requests an operation on a CDS name, ACL management software on a server that stores the name examines the ACL entries associated with the name. The software then grants or denies the operation based on the permissions granted to the requesting principal in the ACL entries. Similarly, when a principal requests a privileged operation on a CDS clerk or server, ACL management software on that system examines the ACL entries that are associated with the principal name that represents the clerk or server. The software then grants or denies the operation, based on the permissions granted to the requesting principal in the ACL entries. The DCE control program (**dcecp**) provides commands that add, modify, copy, delete, and display ACLs that are associated with CDS names, clerks, and servers. See the *DCE for AIX Administration Command Reference* for detailed information on commands. The remainder of this section describes DCE authorization as it applies specifically to CDS. Before you try to create or modify permissions to CDS names, clerks, or servers, read about the cell directory service in this guide for complete information on the DCE authorization mechanism.

Overview of the Merge and Append Procedures

Introduction — Overview of Merge and Append Procedures

To merge or append CDS directories, you use the DCE control program's (**dcecp**) **directory merge** command.

Steps — Overview of Merge and Append Procedures

The basic steps for both procedures are:

1. At your system prompt, enter **dcecp** to invoke the DCE control program.
2. Merge or append one existing directory with another existing directory. To do this, use the **directory merge** command to combine the directory's information about its descendants (object entries, soft links, and child directories) with another directory's information or to append the information below an existing bottom-level directory.
3. Delete the source directory or subtree (and its contents) that you merged in Step 2 from its old location in the hierarchy by using the **directory delete** command. Replace the deleted directory information with a single soft link of the same name to redirect lookups of the information at the new location by using the **link create** command.

Note: The presence of clearinghouses, duplicate names, or unreachable names in a merged directory requires special handling. The merge and append operations described in the following sections assume that no duplicate names exist in the source and target directory or subtree, and that the clearinghouses that store the master replicas of affected directories are enabled and reachable at the time the operations are initiated.

Example — Overview of Merge and Append Procedures

The example merge and append operations described in this section are based on an example namespace, shown in the following figure.

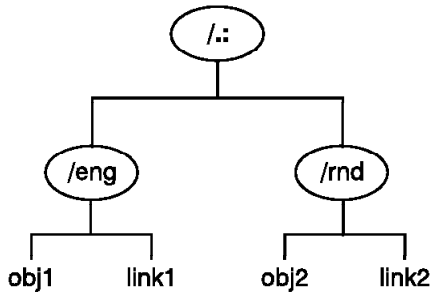


Figure 5-16. An Example of Namespace Hierarchy

The example namespace consists of two directories under the root — /**.:/eng** and /**.:/rnd**. The source directory (/**.:/eng**) contains two entries — /**.:/eng/obj1** and /**.:/eng/link1**. The target directory (/**.:/rnd**) also contains two entries — /**.:/rnd/obj2** and /**.:/rnd/link2**.

Preserving a Clearinghouse Across a Server System Upgrade

Introduction — Preserving a Clearinghouse

If you plan to upgrade the operating system software on a CDS server system, and you want to preserve the clearinghouse (or clearinghouses) on the system, use the steps described in this section.

Steps — Preserving a Clearinghouse

1. Make sure that you disable the clerk and server.
2. Before you perform the system upgrade, back up the following CDS files:

   cds_attributes
   cds_files
   _ch.checkpointnnnnnnnn***
   _ch.tlognnnnnnnn***
   ***_ch.version**
   cds_cache.*nnnnnnnn***
   cds_cache.version
   cds_cache.wan

   (where files ending with *nnnnnnnnn* represent a file suffix). See the *DSS for AIX Up and Running!* for the full path names of all CDS files.
3. Perform the system upgrade.
4. Restore all the files that you backed up in Step 2.
5. Follow the procedure described earlier in this section for restarting a server. When the server process starts, it automatically locates the appropriate restored files and starts all clearinghouses on the system.

Propagating Permissions to CDS Directories and Their Contents

Introduction — Propagating Permissions

By creating all three ACL types (Object ACL, Initial Object Creation ACL, and Initial Container Creation ACL) for a directory, you can grant access not only to the directory itself but also to the directory's future contents and all child directories (and their contents) that may later be created.

More Detailed Information — Propagating Permissions

If you just created a new directory named /**.:/sales** and you create an ACL entry of the Object ACL type that grants user **Smith** read permission to the /**.:/sales** directory, **Smith** can:

- Read the attributes associated with the /**.:/sales** directory, and
- Display the names stored in the /**.:/sales** directory.

If you create a second ACL entry of the Initial Object Creation ACL type that grants **Smith** read permission to the /**.:/sales** directory, **Smith** can:

- Read the attributes associated with the /**.:/sales** directory,
- Display the names stored in the /**.:/sales** directory, and
- Read the attributes associated with all the names that you may later create in the /**.:/sales** directory, unless prohibited by explicit ACL modification after their creation.

If you create a third ACL entry of the Initial Container Creation ACL type that also grants user **Smith** read permission to the /**.:/sales** directory, **Smith** can:

- Read the attributes associated with the /**.:/sales** directory,
- Display the names stored in the /**.:/sales** directory,
- Read the attributes associated with all the names that you may later create in the /**.:/sales** directory, and
- Perform all of the three preceding operations on all child directories that may later be created under the /**.:/sales** directory.

See *DCE for AIX Administration Command Reference* for complete information on default ACLs.

Relocating a Clearinghouse

Introduction — Relocating a Clearinghouse

Note: This section describes the procedure that you use to temporarily relocate a clearinghouse from one CDS server to another. You cannot use this procedure to configure additional CDS server systems. See *DSS for AIX Up and Running!* for information on how to configure CDS servers and CDS clerks.

Occasionally you may need to relocate a clearinghouse from the server system where it currently resides to another server system. For example, you may want to move a clearinghouse when:

- You need to temporarily disconnect the host server system from the network for repair or for other reasons;
- You no longer want the current host system to function as a CDS server; or
- You want to move the clearinghouse to a server system that is physically closer on the network to the user groups and applications that use the information contained in the clearinghouse.

See “Steps — Relocating a Clearinghouse” on page 5-59 for a description of the basic steps used to relocate a clearinghouse.

More Detailed Information — Relocating a Clearinghouse

Disassociating a Clearinghouse from Its Host Server System

Whenever a CDS server starts, one of the tasks the server software performs is to start its clearinghouse (or clearinghouses). The server performs this task automatically by examining a list of the clearinghouses that are resident on the system. Before you relocate a clearinghouse, use the **dcecp** program's **clearinghouse disable** command to update the clearinghouse files and ensure that the files are consistent before you copy them to the target server. The **clearinghouse disable** command also removes, from the source server's internal memory, knowledge of the clearinghouse that you specify. This ensures that the relocated clearinghouse is not automatically started at the source server during server restarts.

After the **clear clearinghouse** command has completed, stop the CDS server with the **dce.clean** command.

Copying the Clearinghouse Database Files to the Target Server System

After you disable the clearinghouse and remove knowledge of the clearinghouse from the host server, you must copy the clearinghouse database files to a specific location on the new host server system.

A clearinghouse database consists of the following three files:

- *clearinghouse-name***.checkpoint***nnnnnnnn*
- *clearinghouse-name***.tlog***nnnnnnnn*
- *clearinghouse-name***.version**

where:

clearinghouse_name	Represents your clearinghouse name.
nnnnnnnn	Represents an 8-digit number.

Verify the existence of these files before you attempt to copy them to the new host system. See *DSS for AIX Up and Running!* for the full path names of all CDS files.

Note: You may sometimes find two **.checkpoint***nnnnnnnn* files in the directory. This can happen as a result of a system crash or other interruption during the most recent clearinghouse checkpoint operation. If you do find two files, copy both of them to the target server system. The server software on that system automatically reconciles any problem that may exist as soon as the clearinghouse is enabled at the target server.

To move the database files to the new CDS server, use the **ftp** utility or a similar network file transfer utility. Copy the three database files from the existing server host to the new CDS server host. The directory where the files reside on the old and new CDS server is **/opt/dcelocal/var/directory/cds**.

Starting the Clearinghouse on the Target Server

After copying the clearinghouse database files to the appropriate location on the target server system, use the **clearinghouse create** command to start the clearinghouse at the new location. Make sure that you specify the same clearinghouse name that was used at its original (source) location. After you enter the command, the server detects the clearinghouse files, adds knowledge of them to its memory, and then starts the clearinghouse.

Steps — Relocating a Clearinghouse

To relocate a clearinghouse, follow these general steps:

1. Disassociate the clearinghouse from the server where it is currently running.
2. Copy the clearinghouse database files from their current location (source server system) to their new location (target server system).
3. Create a new clearinghouse on the target server system by using the same name that was used on the source server system from which you copied the database files.

Permissions Required — Relocating a Clearinghouse

- To use the **clearinghouse disable** command, you must have write permission to the server on which the clearinghouse resides.
- To use the **clearinghouse create** command for the purpose of relocating a clearinghouse, you must have write permission to the server on which you intend to relocate the clearinghouse.

Examples — Relocating a Clearinghouse

In this example, the clearinghouse is named /**.:/my_ch**, where *d* is the drive letter on which the **dcelocal** directory resides.

1. Run the **dce_login** as **cell_admin** on CDS servers **A** and **B**.
2. The following command removes knowledge of clearinghouse /**.:/my_CH** from the memory of its host server. First, server **A**:

   ```
   dcecp> clearinghouse disable /.:/my_ch
   ```

3. **cd /opt/dcelocal/var/directory/cds**
4. Look for files associated with the clearinghouse called /**.:/my_ch**. For example:

   ```
   acell#my_ch.checkpoint00000001
   acell#my_ch.tlog00000001
   acell#my_ch.version
   ```

   Transfer the files found to /**opt/dcelocal/var/directory/cds** on server **B**.
5. Delete the files associated with the clearinghouse you are relocating from server **A**.
6. The following command, which was issued on server **B**, relocates the clearinghouse:

   ```
   dcecp > clearinghouse create /.:/my_ch
   ```

Restarting Clerk and Server Processes

Introduction — Restarting Clerks and Servers

CDS clerk and server processes are created and started automatically by start-up scripts that execute whenever the host system is rebooted. Sometimes, however, you may need to run these scripts yourself if a clerk or server fails to start automatically upon reboot, or if you want to restart a clerk or server that you disabled to perform a backup or do diagnostic work on the host system.

Steps — Restarting Clerks and Servers

- To restart a clerk:
  1. Log into the clerk system as superuser (**root**).
  2. Enter the following command to determine if the **dced** process is already running:

     ```
     # ps -e
     ```

  3. If the **dced** process displays as an active process, proceed to Step 4. If the **dced** process does not appear on the list of active processes, enter the following command:

     ```
     # dced
     ```

  4. Enter the following command to start the **cdsadv** process:

     ```
     # cdsadv
     ```

- To restart a server:
  1. Log into the server system as a superuser (**root**).
  2. Enter the following command to determine if the **dced** process is already running:

     ```
     # ps -e
     ```

  3. If the **dced** process displays as an active process, proceed to Step 4. If the **dced** process does not appear on the list of active processes, enter the following command to start the process:

     ```
     # dced
     ```

  4. Enter the following command to determine if the **cdsadv** process is already running:

     ```
     # ps -e
     ```

  5. If the **cdsadv** process displays as an active process, proceed to Step 6. If the **cdsadv** process does not appear on the list of active processes, enter the following command to start the process:

     ```
     # cdsadv
     ```

  6. Enter the following command to restart the server:

     ```
     # cdsd
     ```

When the server process starts, all clearinghouses on the system also start.

Restoring Merged CDS Directories

Introduction — Restoring Merged CDS Directories

You can use the **dcecp** program's **link delete** and **directory merge** commands to restore deleted directories and their contents to your namespace.

More Detailed Information — Restoring Merged CDS Directories

First run the **link delete** command to remove the soft links in the former source location, then use the **directory merge** command to append the copy of the directory back under its former parent directory.

If the directory has slave replicas, use the **directory create** command to create a new replica of the directory in each of the clearinghouses from which the directory was deleted.

Remember that the **directory merge** command affects only directories and their contents. It does not copy clearinghouses or their associated clearinghouse object entries and therefore cannot be used to

restore clearinghouses or to account for discrepancies in information among individual replicas resident on different clearinghouses. Furthermore, the directory information in a particular location may have changed since the time of the original merge operation.

Security in the Cell Directory Environment

Introduction — Security in Cell Directory Environment

In a secure DCE cell operation, a server does not complete a user's request unless the user's identity has been verified through the DCE Authentication Service. For example, a CDS server allows a user to create a new directory only if that user's identity has been verified. The process of verifying that users are who they say they are is called *authentication*. The proof is in the form of a user name or principal name coupled with a special kind of password.

More Detailed Information — Security in Cell Directory Environment

CDS servers themselves must be authenticated principals for two reasons:

- To prove to clients that they are trustworthy, and
- To prove to each other that they have the permission to modify and manage the data that they share.

The principal name of a CDS server is automatically selected by the configuration program and is placed in a group that contains the names of all CDS servers in the cell. The group is stored as an entry in the DCE Security Service database. After initial contact with a CDS server, the clerk confirms through the Security Service that the server is a valid member of the server group.

Authentication is not an end in itself, but is actually a step in the process of authorization. After the identity of a principal has been verified, the software must next determine whether that principal has the permissions required to perform a requested action. This is called *authorization*. Therefore, to create a new directory, the user in the previous example must not only be authenticated, but must also have the appropriate permissions.

Servers need to be authenticated to each other because they share and modify replicated data. For example, suppose Server A and Server B both store a replica of the same directory. Associated with each directory is a list of all the servers authorized to maintain that directory. When a user modifies an entry in the replica at Server B, Server B must notify Server A of the change. Server A does not accept the update unless Server B is an authenticated principal and is one of the principals authorized to modify that directory.

Permissions Required — Security in Cell Directory Environment

The CDS permissions are read, write, insert, delete, test, control, and administer. Each has a slightly different meaning depending on the kind of name with which it is associated. In general, their meanings are as follows:

- Read permission lets users view data.
- Write permission lets users add or change data.
- Insert permission lets users create entries in a directory.
- Delete permission lets users delete entries.
- Test permission lets users test whether an attribute of a name has a specific value without being able to see any values (that is, without having read permission to the name). The main advantage of this

permission is that it gives application programmers a more efficient way to check for a value. Rather than reading a whole set of values, the application can test for a particular value.

- Control permission lets users manage the Access Control List (ACL) of an entry.
- Administer permission lets users manage directory replication.

Note: It is possible to define a special ACL for users who cannot be authenticated or who deliberately request unauthenticated operations. In such a case, the user's identity is not verified and the ACL entry for unauthenticated users determines whether the user has the permissions to perform the requested action. See *DCE for AIX Administration Command Reference* for details on creating ACLs for unauthenticated users.

Setting Up Access Control in a New Namespace

Introduction — Setting Up Access Control

You should plan a consistent access control policy and be ready to implement the policy as soon as you configure your first CDS server and *before* you create or populate any new directories. Among the tasks you can perform are:

- Adding members to the namespace authorization group,
- Creating additional authorization groups, and
- Establishing maximum permissions for unauthenticated principals.

More Detailed Information — Setting Up Access Control

Adding Members to the Namespace Authorization Group

To facilitate managing and troubleshooting your namespace, the cell configuration process creates a namespace authorization group under the fixed name **subsys/dce/cds-admin**. The configuration process then grants the group full access to the cell root directory. This access propagates to the entire namespace as it evolves.

Immediately after its creation, the authorization group contains only the name that the initial namespace administrator specified during the cell configuration process. You can use the **dcecp** program's **group add** command to add the principal names of other individuals in your organization whom you want to administer and troubleshoot the namespace. Because this group possesses full access to the entire namespace, its members can intervene, whenever necessary, to solve problems for namespace users with fewer permissions. By removing a user's principal name from the group, the user described by that principal loses the access assigned to the group. See “Maintaining Policies and Properties” on page 1-58 for complete information on how to add and delete group members.

Creating Additional Authorization Groups

Authorization groups can provide a convenient and flexible way to control access to your namespace. You can combine users according to organization, work type, security status, and so on, and then grant each group a specific set of permissions to specific directories or other names in the namespace.

To delegate authority locally, you can create an authorization group for each of the functional directories that you plan to create in your namespace. For example, you could create an authorization group named **subsys/dce/sales-admin** and include, as members, the individuals who are responsible for managing the /**.:/sales** directory. Each local authorization group could have full access to the contents of the directory for which it is responsible.

Establishing Maximum Permissions for Unauthenticated Principals

If you want to apply a namespace-wide set of maximum permissions for all unauthenticated principals, you should do so immediately *after* you configure your first CDS server and *before* you create and populate any directories below the cell root. By creating an unauthenticated ACL entry and an **any-other** entry for the cell root by using the Object ACL and Initial Container Creation ACL types, you can take advantage of automatic propagation of the unauthenticated entry to the entire namespace as it evolves.

Skulk Operation

Introduction — Skulk Operation

The skulk operation is a periodic distribution of a collection of updates. Its main functions are to ensure that replicas receive changes that might not have reached them during an update propagation and to remove outdated information from the namespace.

More Detailed Information — Skulk Operation

CDS skulks each directory individually. During a skulk, CDS collects all changes that were made to the master replica since the last successful skulk and then disseminates the changes to read-only replicas of the directory. All replicas must be available for a skulk to be considered successful. If CDS cannot contact a replica, it continues making changes in the replicas that it can contact and generates an event to notify you of the replica or replicas it could not update. CDS then periodically reattempts the skulk until it completes successfully.

A skulk can begin in one of three ways:

- A CDS manager can enter a command to start an immediate skulk on a directory.
- CDS starts a skulk as an indirect result of other namespace management activities, which include:
  - Adding or removing a replica,
  - Creating or deleting a directory,
  - Redesignating replica types, and
  - Adding or deleting a child cell name in a parent cell.

  All of these activities produce changes in the structure of the namespace, so an immediate skulk ensures that the new structure is reflected throughout the namespace as quickly as possible.
- The CDS server initiates skulks automatically at a routine interval called the *Background Skulk Time*.

  The Background Skulk Time interval guarantees a maximum lapse of time between skulks of a directory regardless of other factors, such as namespace management activities and user-initiated skulks. A CDS server periodically checks each master replica in its clearinghouse and initiates a skulk if changes were made in a directory since the last successful skulk of that directory.

Restrictions — Skulk Operation

If you use the **cdscp clear clearinghouse** or **dcecp clearinghouse disable** commands to disable a clearinghouse on a CDS server, skulks may fail until the clearinghouse is re-enabled. Until the clearinghouse is enabled, you may receive messages indicating that DCE is unable to communicate with any CDS server. Clearinghouses should only be disabled temporarily while maintenance is being performed. Use the **cdscp delete clearinghouse** or **dcecp clearinghouse delete** commands if the clearinghouse is to be removed completely.

Skulking a Directory

Introduction — Skulking a Directory

The skulk operation is a periodic distribution of recent modifications that were made to the namespace. CDS skulks every directory at regular intervals according to the value assigned to the directory's **CDS_Convergence** attribute. To ensure that updates are distributed to all replicas of a directory as soon as possible, you can initiate a skulk of the directory by using the **directory synchronize** command rather than waiting for the next scheduled skulk to distribute the new information. You can use this command to perform the following tasks:

- Distribute crucial updates that were made to a directory's contents or replica set when you do not want to wait for the next skulk.
- Skulk the directories that store replicas on servers that were inoperative for an extended period and were just brought back online.

More Detailed Information — Skulking a Directory

Entering the directory synchronize Command

Use the **directory synchronize** command to initiate an immediate skulk on the directory that you specify.

After you enter the command, the **dcecp** program temporarily suspends the `dcecp>` prompt while the skulk is in progress. Skulks of directories with large replica sets may take some time to execute. If the prompt returns with no error messages, the skulk is successful. If error messages are displayed before the prompt returns, the skulk failed.

For a skulk to succeed, every replica in the directory's replica set must be reachable. Skulks may sometimes fail when directories have large replica sets or when the servers that store replicas of the directory are located over great distances where network connectivity is not always reliable.

Skulk failure does not make CDS unusable. Although the skulking process is unable to update information in a replica that it cannot contact, it always updates information in the replicas that it can reach. Temporarily, some replicas contain the latest information and some do not. When a skulk fails, CDS automatically repeats the skulking process, at an interval based on the directory's convergence value, until all replicas in the set are updated with the latest changes. When all replicas contain identical information, CDS considers the skulk successful.

If skulks of a particular directory continue to fail, you can determine the cause by reviewing the log of CDS events on the server that stores the master replica of the directory.

Synchronizing CDS Server Clerks

After performing a skulk operation on a directory, you may receive the message `Server clocks are not synchronized`, indicating that the server clocks are not synchronized. If so, first check to see whether the system clocks on the server systems are indeed synchronized. If they are and you still receive the message, then perhaps the system clock on an individual server was mistakenly set to a future time and subsequently restored. This causes a problem for CDS because there may be timestamps stored in a clearinghouse that are invalid (any timestamp greater than five minutes in the future from the current time).

If this is the case, adjust the system clock to the current time and enter the following command:

`dcecp>` **`clearinghouse repair`** *`clearinghouse-name`* **`-timestamps`**

This command disables the clearinghouse, analyzes and repairs bad timestamps, checkpoints the clearinghouse to disk, and re-enables the clearinghouse. To use the command, you need write permission to the server on which the clearinghouse resides. Also, you should execute this command on all clearinghouses that replicate the directory (and its objects) that needs to be repaired.

After executing the **clearinghouse repair** command, you should be able to skulk the directory successfully.

Permissions Required — Skulking a Directory

To skulk a directory, you need:

- Administer, write, insert, or delete permission to the directory, and
- The server principal for the server system where you enter the **directory synchronize** command needs read, write, and administer permissions to the directory that you intend to skulk.

  If the server is included in the server authorization group **subsys/dce/cds-servers**, these permissions should already be in place. If in doubt, use the **acl show** command to verify that the server principal has the appropriate permissions. See *DCE for AIX Administration Command Reference* for complete information on using the **acl show** command.

Example — Skulking a Directory

The following command initiates a skulk on the /**.:/admin** directory:

```
dcecp> directory synchronize /.:/admin
```

Translating from Names to Resources

Introduction — Translating from Names to Resources

In the same way directory names in a logical namespace hierarchy translate to physical replicas in clearinghouses, CDS names translate to physical resources that are used either internally by CDS or by client applications. The attributes of a name are what make the translation possible. This section describes the relationship between CDS names and the physical resources they describe.

More Detailed Information — Translating from Names to Resources

To discover the physical location of a resource, CDS looks up an attribute associated with its name. The figures in “Examples — Translating from Names to Resources” on page 5-66 illustrate the connection between the various kinds of CDS names and the resources that they describe. The figures are based on the namespace in Figure 5-17. All of the names in the last three figures are in the same cell namespace, as evidenced by the use of the /**.:** prefix to represent the cell root. See “Managing Intercell Naming” on page 1-31 for information about name resolution across multiple cells.

Figure 5-18 on page 5-67 shows the relationship between two clearinghouse object entries and the clearinghouses they describe. A clearinghouse object entry differs from other kinds of object entries in that it is created, used, and maintained by the CDS software instead of by a client application. However, it is like any other object entry in that it describes a physical resource in the network or clearinghouse. CDS creates the object entry automatically when you create and name the clearinghouse.

Figure 5-20 on page 5-69 shows the relationship between directories and their associated child pointers. It illustrates that, although a child pointer has the same name as its associated directory, the child pointer is a separate entry in the namespace and resides in the parent of the directory to which it refers.

Examples — Translating from Names to Resources

- Figure 5-17 shows three directories and their contents in a logical namespace and illustrates how replicas of those directories are physically implemented in two clearinghouses. The clearinghouses themselves have CDS names — /**.:/Paris_CH** (on Node 1) and /**.:/NY_CH** (on Node 2). The **_CH** suffix is a recommended convention for naming clearinghouses. The /**.:/Paris_CH** clearinghouse contains replicas of the root directory and the /**.:/subsys**/**PrintQ** directory. The /**.:/NY_CH** clearinghouse contains replicas of the root directory and the /**.:/subsys** directory. Recommended practice is to create at least two replicas of every directory. Therefore, the /**.:/subsys** and /**.:/subsys**/**PrintQ** directories each need to be replicated in at least one other clearinghouse somewhere in the cell.

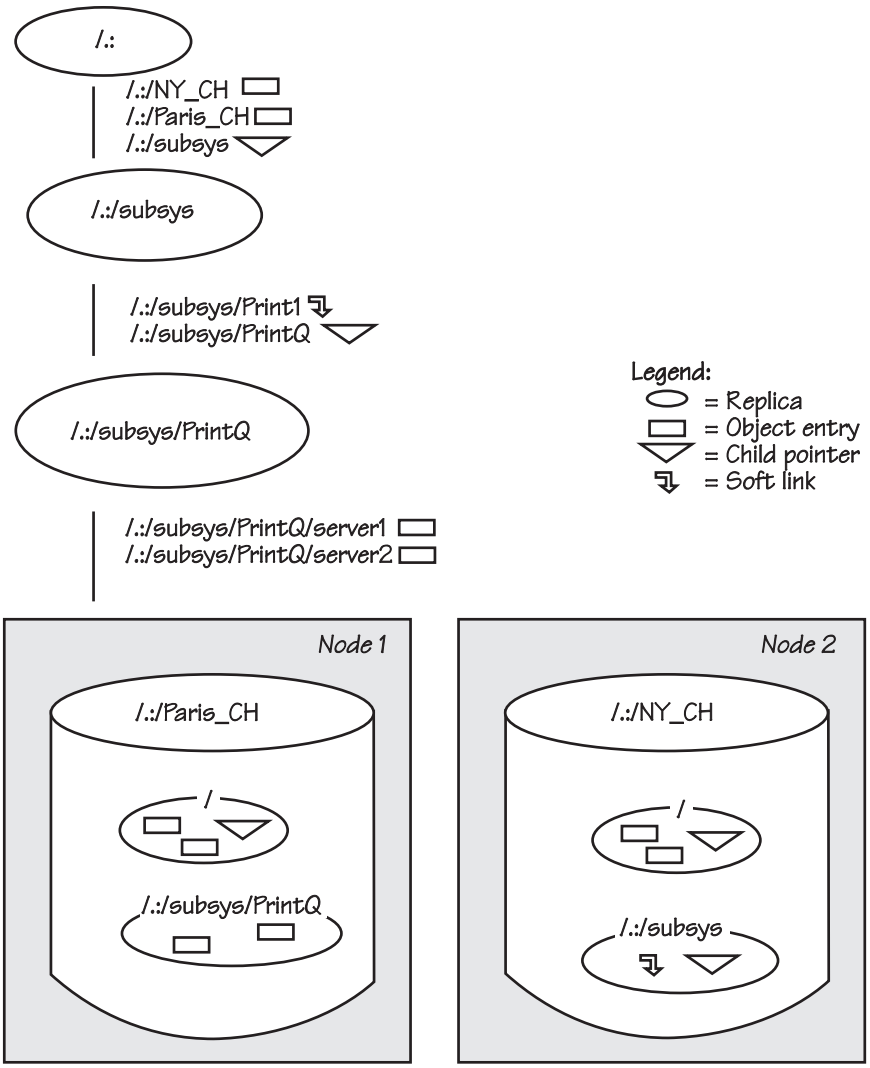


Figure 5-17. Logical and Physical Views of a Namespace

- Figure 5-18 on page 5-67 shows two clearinghouse object entries — /**.:/Paris_CH** (which points to the clearinghouse that is named /**.:/Paris_CH** on Node 1) and /**.:/NY_CH** (which points to the

clearinghouse that is named **/.:/NY_CH** on Node 2). Each clearinghouse object entry has a **CDS_CHLastAddress** attribute, whose **Tower** subattribute contains RPC binding information that CDS uses to contact the node where the clearinghouse resides. See “Object Identifier Files” on page 1-59 for a list of CDS attributes and their descriptions.

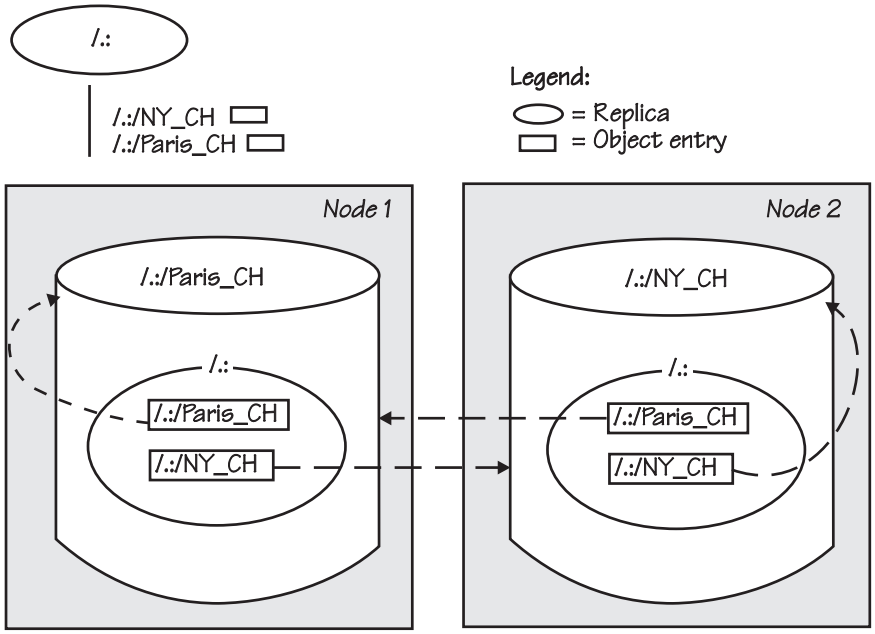


Figure 5-18. Clearinghouse Object Entries and Clearinghouses

- Figure 5-19 on page 5-68 shows the relationship between a soft link, the object entry it points to, and the resource that the object entry describes. The soft link, **/.:/subsys/Print1**, has an attribute called **CDS_LinkTarget**, which contains the name to which that link points (an object entry named **/.:/subsys/PrintQ/server1**). The object entry describes a print server machine that is used by an application called **PrintQ**. The replica containing the **/.:/subsys/PrintQ/server1** object entry exists in the **/.:/Paris_CH** clearinghouse. The object entry has a **CDS_CHLastAddress** attribute whose **Tower** subattribute contains RPC binding information that enables the **PrintQ** application to contact the print server machine.

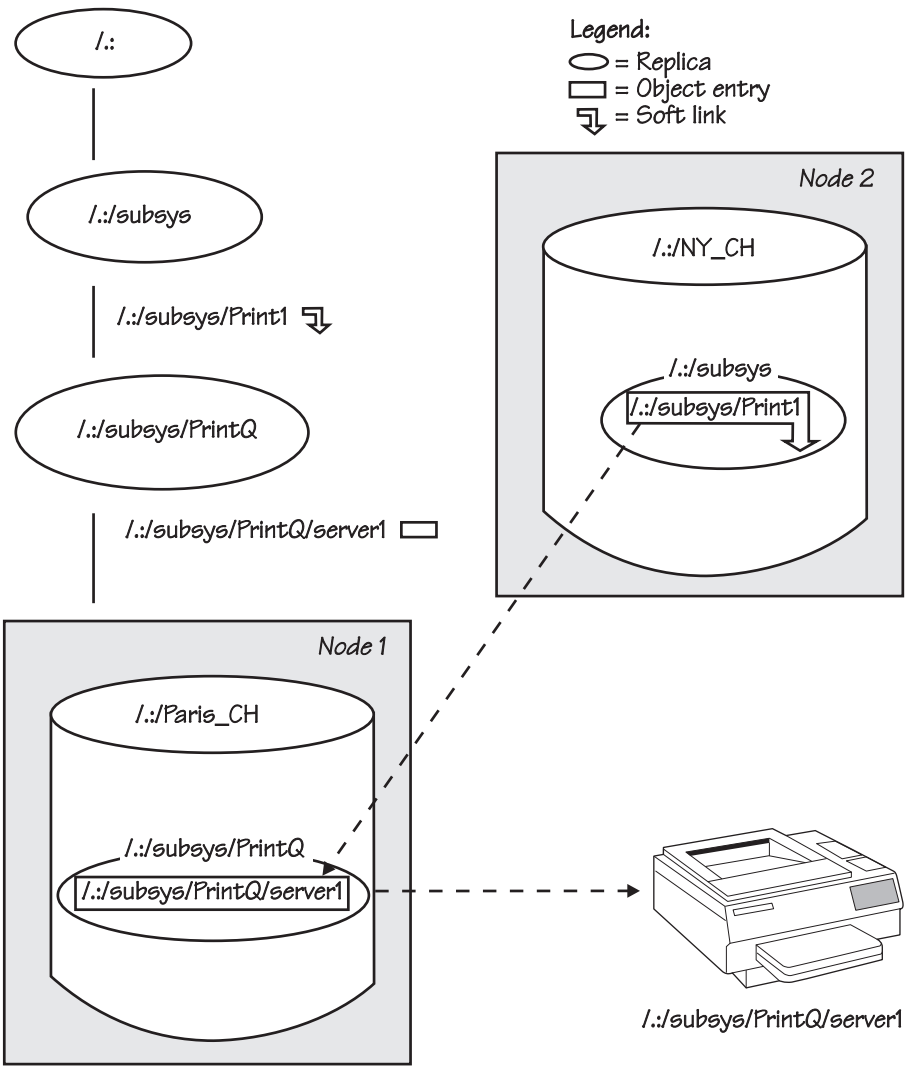


Figure 5-19. A Soft Link and Its Resolution

- The root replicas in both clearinghouses contain a child pointer for the /**.:/subsys** directory. The /**.:/subsys** child pointer has an attribute called **CDS_Replicas**, which contains the name and address of the /**.:/NY_CH** clearinghouse where a replica of the /**.:/subsys** directory exists.

  In the /**.:/NY_CH** clearinghouse, the replica of the /**.:/subsys** directory contains a child pointer for the /**.:/subsys/PrintQ** directory. The child pointer's **CDS_Replicas** attribute contains the name and address of the /**.:/Paris_CH** clearinghouse, where a replica of the /**.:/subsys/PrintQ** directory exists.

  When a directory has multiple replicas, as is normally the case, the **CDS_Replicas** attribute lists all of the clearinghouses containing a replica of the directory. You can use the DCE control program's **directory show** command and clearinghouse options to display this attribute.

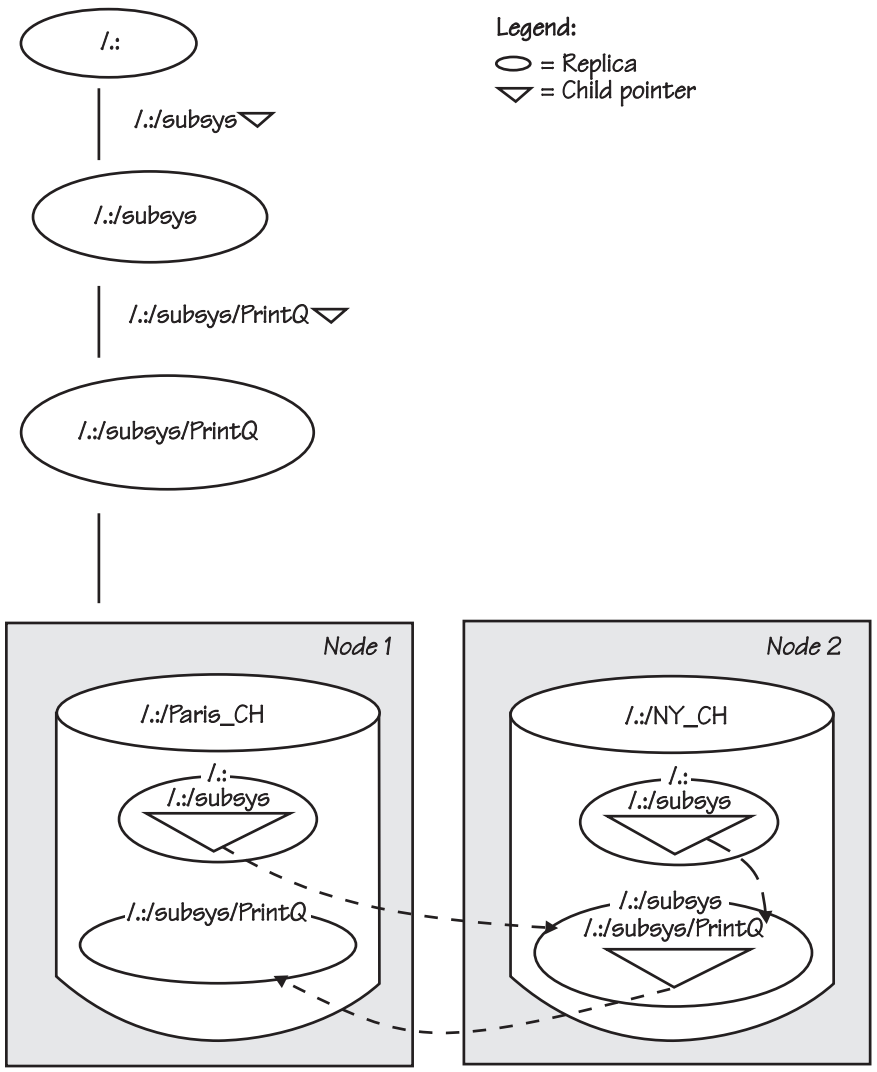


Figure 5-20. Child Pointers and Directories

Update Propagation

An update propagation is an immediate attempt to apply one change to all replicas of the directory in which the change was just made. Its main benefit is that it delivers each change in an efficient and timely manner. Unlike a skulk operation, however, update propagation does not guarantee that the change gets made in all replicas. If a particular replica is not available, the update propagation does not fail but the change is not made in that replica. The skulk operation ensures that, when the replica is available again, it becomes consistent with the other replicas in its set.

You can tune the degree of persistence that CDS uses in attempting an update propagation (or prevent propagation) by adjusting a directory attribute called **CDS_Convergence**. Convergence also affects the frequency of skulks on a directory. See “Managing CDS Directories” on page 1-30 for details on viewing and changing a directory's convergence.

Chapter 6. DCE Distributed Time Service

This section describes clock synchronization processes and related tasks. For introductory and overview information on the DTS, see “Distributed Time Service (DTS) Overview” on page 1-34. That section also groups related tasks together in the order in which they may normally occur so you can access these referenced tasks in a logical, tutorial-type manner. You may find this helpful in performing related tasks together from beginning to end.

Changing the System Time

Introduction — Changing the System Time

There are three ways you can use **dcecp** program commands to change the system time. This section provides reasons for changing the system time and show examples of the commands that you can use to modify the time and change the system clock.

More Detailed Information — Changing the System Time

Updating the Time Monotonically

If your network does not use time providers and the network systems have been running for some time, you may want to update the time on several systems to match UTC or another external reference. When time providers are absent from your network, the systems remain closely synchronized but their clocks may drift away from accepted time standards such as UTC.

Use the **dcecp** program's **clock set** command to modify the time on a server system to make it more accurate. The DTS synchronization process ensures that the new time you supply with the command is propagated to the other network systems. In order to update the system clock to a new time, the new time and inaccuracy you specify for a system must be a smaller interval than the current system interval.

In order to use the **clock set** command effectively, you must have temporary access to a trusted time reference. Such references can include the time signals that many standards organizations disseminate by radio or telephone. You can also use a clock that you have recently verified as accurate. See “Time Providers and Time Services” on page 1-60 for more information.

Because it is a manually-entered command that is used to modify an absolute time, the **clock set** command is not useful for small inaccuracy settings. The minimum reliable inaccuracy that you can achieve with the command is approximately 1 second. Human error and processing delays combine to make lower settings unreliable. For example, you enter the command and the new time and then begin monitoring the reference. When you perceive that the reference has reached the desired time, you press <Enter> to initiate the command. Your perception of the reference mark and your pressing <Enter> do not exactly coincide. Furthermore, after the command is initiated, DTS takes time to interpret and execute the command.

If your systems require synchronization that is closer than 1 second to a standard such as UTC, consider purchasing one of the time providers listed in “Time Providers and Time Services” on page 1-60. All of the time providers described in the listing compensate for transmission and processing delays and can provide time references that are accurate to the millisecond level.

Updating the Time Nonmonotonically

Use the **clock set** command with the **–abruptly** option when you want to abruptly set the time for a server system. The command immediately (nonmonotonically) changes the system clock setting to the specified time, rather than gradually (monotonically) adjusting the time.

Note: Use caution when changing the system time abruptly. The abrupt adjustment of the time is appropriate at system startup or when the system clock is faulty and you identify and correct the problem. Changing the system time to a setting that falls outside the time intervals of the system's known servers causes DTS to declare the system faulty at the next synchronization.

Because the **clock set** command is generally used to correct gross clock errors, it is likely that the time you specify for a given system will appear faulty to the system's known servers if the system and servers have the same epoch number. You *can* prevent the systems whose times you are changing from being declared faulty. Use the **clock set** command's **–epoch** option, along with the **–abruptly** option, to set the new time to isolate it from the other systems. You can then change the time and epoch for the other systems until all the systems share the same epoch again. This process is useful in the rare case when the majority of servers in the network are faulty.

In order to use the **clock set** command effectively, you must have temporary access to an accurate time reference. Such references can include the time signals that many standards organizations disseminate by radio or telephone. You can also use a clock that you have recently verified as accurate. See “Time Providers and Time Services” on page 1-60 for a list of time sources.

Because it is a manually-entered command that is used to modify an absolute time, the **clock set** command is not useful for small inaccuracy settings. The minimum reliable inaccuracy that you can achieve with the **clock set** command is approximately 1 second. Human error and processing delays combine to make lower settings unreliable. For example, you enter the command with the new time and then begin monitoring the reference. When you perceive that the reference has reached the desired time, you press <Enter> to initiate the command. Your perception of the reference mark and your pressing of the <Enter> key do not exactly coincide. Furthermore, after the command is initiated, DTS takes time to interpret and execute the command.

Forcing System Synchronization

After you create and enable DTS on all systems in your network, they synchronize without any further intervention. There are situations, however, when you may want to force a system to synchronize immediately rather than waiting the amount of time specified by the **syncinterval** and **maxinaccuracy** attributes. As an example, you may want to synchronize a system with a TP server that you have just added to the network.

To forcibly synchronize the clock on a system, use the **dts synchronize** command. If you enter the **dts synchronize** command without the optional **–abruptly** option, the time is adjusted gradually. If you enter the **dts synchronize** command with the **–abruptly** option, the time is immediately adjusted. In the situation posed by our example, you might want to use the command with the **–abruptly** option to have the narrow time interval contributed by the time provider quickly propagated throughout the network:

```
dcecp> dts synchronize -abruptly
```

Examples — Changing the System Time

- The following example shows how to monotonically update the time on a server system (that is, how to reset the clock and eventually propagate the adjustment throughout the network).

  ```
  dcecp> clock set 1993-10-07-09:30:15.00I01.00
  ```

- The following example shows how to change both the time and epoch for a system.

```
dcecp> clock set 1993-10-07-09:30:15.0000I01.0000 -abruptly -epoch 1
```

Configuring DTS for a LAN

If your nodes are in a single local area network, regardless of the number of nodes, planning your DTS implementation is relatively simple. To detect faulty time servers, configure at least three systems as servers. If you want to provide redundancy for your DTS implementation, plan to install four or more servers in the network. That way, if one of the servers fails, DTS can still synchronize with reliable results.

To ensure the reliability of your DTS implementation, you must have stable network connections between server nodes. If you plan to add WAN links to your LAN, do not move the servers to the remote nodes; WAN links are usually less reliable than the LAN.

If you have a single LAN, the location of the servers on the LAN is not critical. You can locate one of the servers on a readily accessible node to aid in trouble shooting, but there are no other recommended server locations. Neither global servers nor couriers are required.

If you are planning to use one or more time providers, locate them at easily accessible systems to ease startup and maintenance. If your network only requires synchronized clocks, but does not need to closely follow a time standard such as UTC, you may not require a time provider. If you do not use a time provider, we recommend that you use the **dcecp** program's **clock set** command to manually set the time approximately once each week.

Figure 6-1 shows a simplified LAN configuration. Your LAN may be much larger, but the figure should resemble a portion of your network.

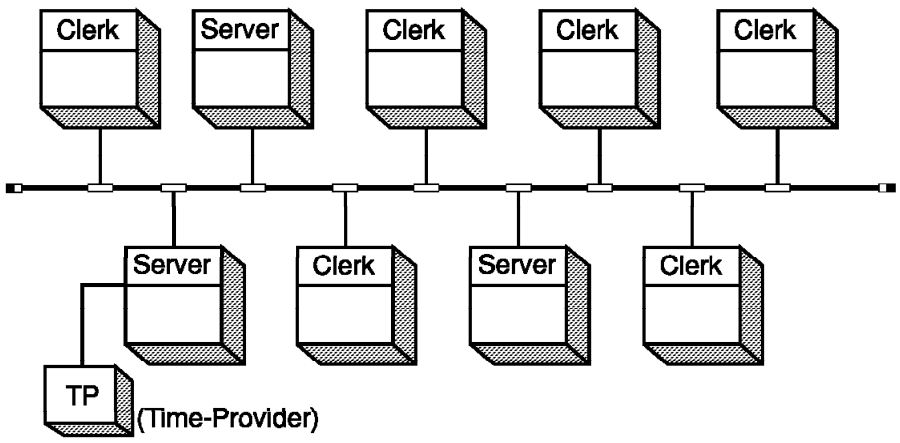


Figure 6-1. DTS Configuration — LAN

Configuring DTS for an Extended LAN

If you follow these guidelines, you can spread the time servers over the network at whatever locations are convenient, just as you would for a single LAN. You need not set up three servers on each segment, because multicast synchronization messages will be relayed over the network interface without delay. When placing the servers (including name servers), you need only consider the quantity of servers, not their locations.

If your network consists of several LAN segments connected by bridges, your network is considered an extended LAN. Planning for extended LANs is similar to planning for a single LAN; treat each segment of the extended LAN as though it were a separate LAN. The following guidelines are recommended:

- Create three servers in each segment.
- Use the **dtscp configure** command to configure one server on each segment as a global server.
- Use the **dts modify** command to configure one server on each segment as a courier.
- If you are using time providers, connect them to the global servers.

Controlling Access to DTS

Introduction — Controlling Access to DTS

You can assign privileges that control access to DTS by using a DCE Authorization Service Access Control List (ACL).

More Detailed Information — Controlling Access to DTS

The DTS principal that represents the server on a given system is the primary access control object for DTS. This principal has controlled access by human users and clerk or server processes. The default name that you can use for the DTS object in any **dcecp** program command is /**.:hosts**/*hostname*/**dts-entity**.

The ACL for the DTS server can contain any type of ACL entry that is valid for a principal (human or process) or authorization group to which this principal belongs. See “Using Access Control Lists” on page 1-59 for a discussion of the DCE ACLs facility and descriptions of ACL types and their entries.

To modify any of the entries in the DTS server principal's ACL, you can use the **acl modify** command. Instructions for using this command appear in “Using Access Control Lists” on page 1-59.

Example — Controlling Access to DTS

To display the ACL entries in the DTS server principal's ACL, you can use the **dcecp** program's **acl show** command as in the following example:

```
dcecp> acl show /.:/hosts/Detroit2/dts-entity

{unauthenticated r--}
{user hosts/Detroit2/self rwc}
{group subsys/dce/dts-admin rwc}
{any_other r--}
```

DTS Advantages

Introduction — DTS Advantages

DTS offers all the features normally provided by a time service, but it also has several features that enhance network performance. The following describes these DTS features:

- Applications Support
- External Time-Provider Support
- Manageability

- Quantitative Inaccuracy Measurement

More Detailed Information — DTS Advantages

Applications Support

Operating systems and distributed applications require synchronized time measurements to coordinate their processes. DTS synchronizes the system clocks in a network with each other, and in the presence of an external time provider, to the UTC time standard. Any distributed application that reads the system clock (which is the majority of applications) needs DTS. As the number of distributed applications and systems in a network increases, DTS becomes increasingly vital to process coordination.

There are several types of existing applications that use the synchronized time DTS provides to system clocks. These applications must reference synchronized system clocks in order to coordinate the events that occur throughout the network. Applications use synchronized clocks for the following functions:

- **Event Measurement** — Applications can read the system clock to start and stop timers and to measure the elapsed time between events.
- **Event Reporting** — Applications can read the clock when an event occurs and append a timestamp to the event report.
- **Event Scheduling** — Applications can read the system clock and add a relative time to determine the occurrence of a future event.
- **Event Sequencing** — Applications can determine the order of events by reading the event report timestamps that are derived from the synchronized system clock.

For new applications, DTS provides an Application Programming Interface (API). The API provides routines that new applications can use to obtain and manipulate binary timestamps. The DTS API supports ANSI C language constructs. See the *DCE for AIX Application Development Guide — Core Components* for further information on the DTS API.

External Time-Provider Support

For most networks, it is desirable to synchronize the system clocks with the UTC time standard. Many commercial devices are available for obtaining the UTC time that is provided by standards organizations; these devices receive signals by short-wave radio, satellite, and telephone. If your network or cell is larger than a single LAN, it is recommended that you use at least one external time provider in combination with the DTS software. See “Time Providers and Time Services” on page 1-60 for a list of suppliers of time provider hardware. Sample time provider programs are available online in the **/opt/dcelocal/usr/examples/dts** file.

DTS servers can synchronize with time providers by means of the Time-Provider Interface (TPI), which is described in the *DCE for AIX Application Development Guide — Core Components*. The TPI specifies the communications between the DTS server process and the time-provider process.

When a DTS server attempts to synchronize, it uses the TPI to check for a time provider process. If one is available, the server synchronizes only with the time provider. If no time provider is present, the server synchronizes with other servers in the network.

By using a time provider with a DTS server, you can ensure that the server is closely synchronized with UTC. When other servers request a time from the server with the time provider (the TP server), the TP server's precise time is propagated throughout the network. See “DTS Usage” on page 1-36 for further information about time providers and the server synchronization process.

Manageability

The DTS synchronization functions run as background processes. Little or no input is required from system managers to synchronize system clocks after DTS is initially configured. DTS can be fault tolerant. It prevents malfunctioning clocks from providing the wrong time to other clocks in the network. Occasionally, however, system managers may need to perform the following functions:

- Identify system clock problems.
- Adjust system clocks.
- Change DTS attributes due to varying network conditions.
- Modify system configuration when the network topology changes.

DTS provides a full-featured management interface that allows system managers to adjust system clocks, change the values of the DTS management parameters and add or remove servers from the network.

To aid in solving problems with system clocks, DTS provides event reporting that notifies system operators and managers in the rare event that a system clock is inaccurate or fails to synchronize.

Quantitative Inaccuracy Measurement

Unlike other network time services, DTS uses manufacturer's specifications and direct observation to determine the inaccuracy of system clocks relative to UTC. DTS appends an inaccuracy measurement to each time value that it uses internally. This measurement takes into account cumulative clock error, communications delays, and processing delays. DTS uses combined time and inaccuracy measurements from one or several sources to calculate the most accurate new clock settings for client systems. See “Synchronizing System Clocks” for further information about the DTS synchronization process.

DTS Configuration for WANs and WAN Links

Introduction — DTS Configuration for WANs and WAN Links

Because there are many variations on WAN configurations (especially in combination with LANs and extended LANs), it is impossible to describe every case where a WAN link can be used to disseminate time. This section does not give recommendations for every case involving a WAN link, but describes how you can set up your DTS implementation using several generic configurations as examples.

Because of the variable delay inherent in any WAN link, it is difficult to maintain a consistent skew between clocks on opposite sides of the link. DTS synchronizes clocks across WAN interfaces, but larger inaccuracies occur between the clocks to account for the worst-case transmission delay during each synchronization.

A reliable and robust DTS implementation is important any time WAN links are part of a cell. Because WANs are less reliable than LANs, plan for some redundancy in any DTS implementation involving WAN links. Try to place servers so that there will always be three or more available, even if one of the WAN links goes down.

“More Detailed Information — DTS Configuration for WANs and WAN Links” on page 6-7 gives recommendations for three basic WAN configurations. Your cell may not exactly match any of the configurations, but you can plan your cell by following the recommendations for each example.

More Detailed Information — DTS Configuration for WANs and WAN Links

LANs with WAN Links to Remote Sites

Figure 6-2 on page 6-7 shows a LAN that incorporates several remote nodes by using WAN links.

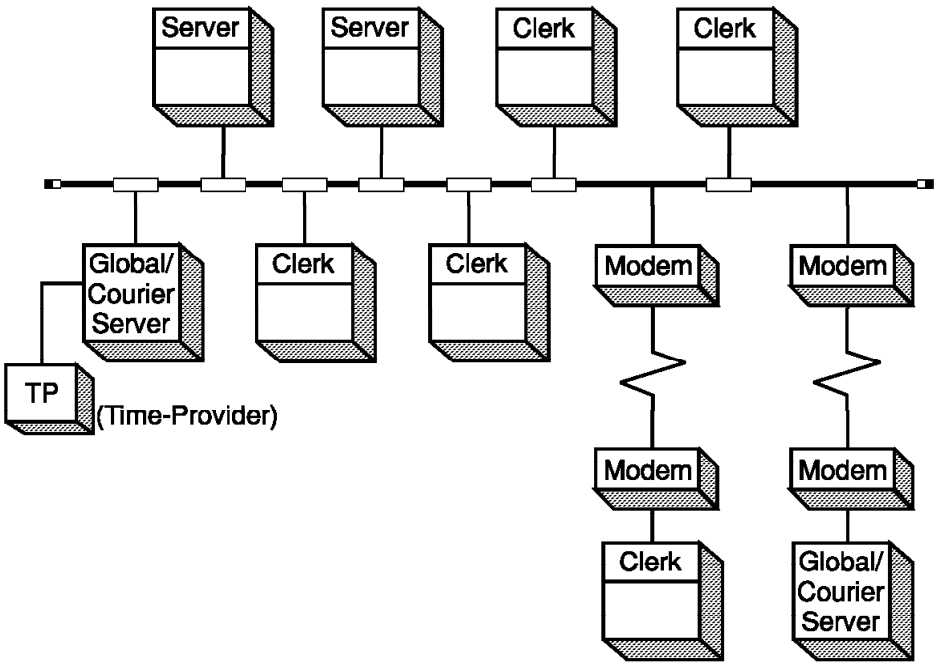


Figure 6-2. DTS Configuration — LAN with WAN Links

In this configuration, follow the basic recommendations for a single LAN, but also adhere to these rules:

- Configure servers at remote sites as global and courier servers.
- The LAN should have a minimum of three servers.
- If you are using a single time provider, locate it at one of the global servers on the LAN rather than at a remote server.

The network configuration resulting from the rules outlined above concentrates the servers on the LAN, so clock skews are kept to a minimum and the service is not dependent on remote nodes that may be physically inaccessible to the system manager. Each remote clerk node synchronizes with the global servers to satisfy the **minservers** attribute setting.

LANs Connected by WAN Links

The rules outlined for extended LANs that use bridges also apply to LANs connected by WAN links. Each LAN in such a network is a separate entity, so several DTS components must be configured on all of the LANs. Configure each LAN according to the following guidelines:

- Configure at least three DTS servers on each LAN.
- Configure at least one server on each LAN as a courier.
- Configure at least one global server on each LAN.
- If you are using time providers, install them at the global servers.

These recommendations will lead to higher DTS efficiency and availability despite the irregular delays associated with WAN links.

WAN Cells

Figure 6-3 shows a geographically-distributed cell that does not have any LANs. DTS delivers higher clock skews in an all-WAN environment than in an all-LAN environment, but still provides synchronization adequate for most distributed applications. In such a network, clock skews are typically less than 5 seconds, but may be as much as 30 seconds if satellite links are used.

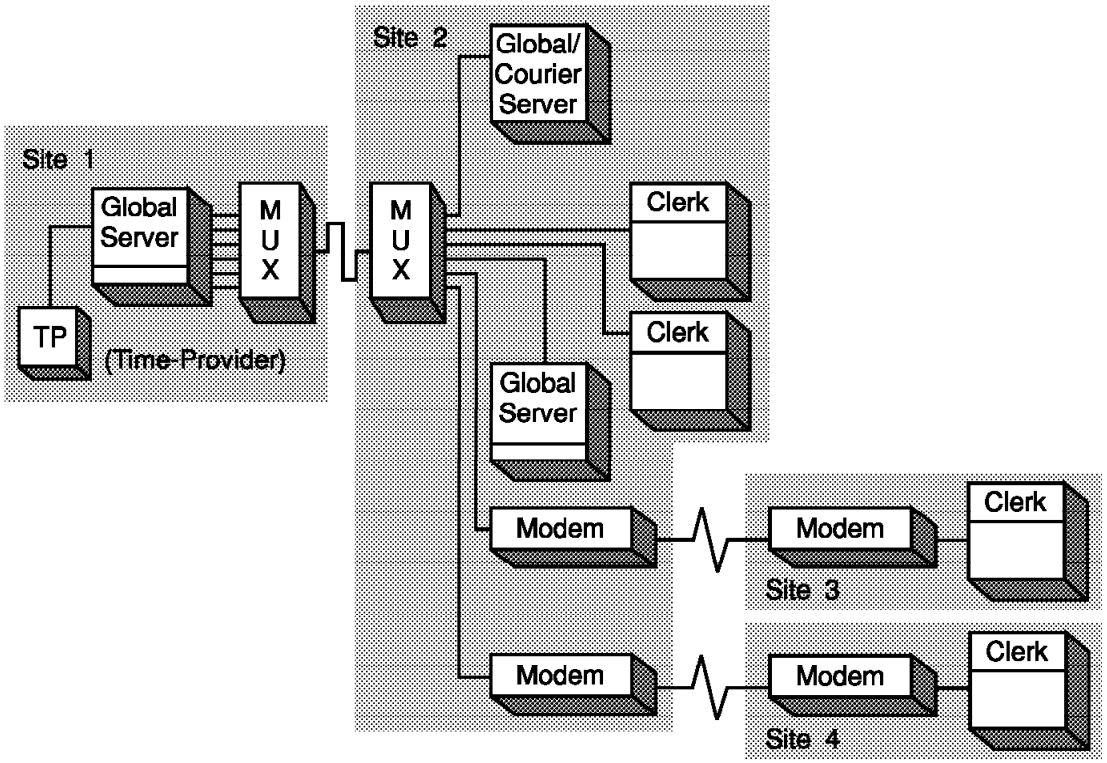


Figure 6-3. DTS Configuration — WAN Networks

Many of the same recommendations for a LAN with WAN links also apply to the network that does not have any LANs. Keep the following considerations in mind when planning your all-WAN network:

- The network should have at least three servers (preferably four or more).
- Every server should be configured as a global server.
- Couriers are not required; however, you can configure any or all of the servers as couriers. The **minservers** attribute will force each global server to synchronize with at least two others.
- You can place the servers anywhere in the network, but place at least one at the central site; choose the most active remote nodes connected by the most reliable links for the rest of the servers.
- If you are using time providers (recommended for this type of network), connect one to a global server node at the central site.

In a geographically-distributed WAN network, also consider ease of access to the nodes by system managers or service personnel. If you locate a server at one of two nodes where traffic patterns and link reliability are equal, locate the server at the node that is convenient to your central site or management facility.

DTS Operations

Introduction — DTS Operations

DTS has two major software components — clerks and servers. "More Detailed Information — DTS Operations" describes each of these components and tells you how they interact to provide time to client applications and to synchronize system clocks.

More Detailed Information — DTS Operations

Clerks

Any system that is not a DTS server is a DTS clerk. Most network systems run clerk software. Clerks maintain server lists and perform the synchronization functions for DTS client systems.

In order to build server lists and synchronize with the servers on the list, clerks need to be able to locate servers automatically. Clerks discover servers by using Remote Procedure Call (RPC) profiles. Profiles are search tables that contain the following types of entries:

- **Server Entries** — The CDS names of individual resource providers
- **Service Group Entries** — A group of resource providers identified by a single CDS name
- **Profile Entries** — The names of other configuration profiles. These entries allow hierarchical nesting of profiles.

Each DTS clerk node contains up to three profiles. When it attempts to locate servers, a clerk first performs an RPC lookup of the entries in a base profile called the *node initial profile*. The clerk then looks for the LAN profile entry. If the LAN profile entry is not found, the clerk searches for the default profile entry. The default profile may contain the LAN profile entry. When the clerk locates the LAN profile, it reads the server entries to build a list of local servers. This process is repeated at set intervals.

If a clerk does not obtain enough server entries as dictated by the DTS management **minservers** attribute, it attempts to locate additional servers, usually those outside the LAN. To locate these servers, a clerk locates the cell profile, which has a well-known CDS name. The cell profile contains global server entries (that is, servers normally found outside the LAN). See "Servers" in this section for further information.

After building a server list with enough entries, a clerk can directly request time values from several of the servers on the list. The clerk then receives these time values and uses them to compute a new system time for its client system.

Servers

Servers provide many of the communications and synchronization functions for DTS. Like clerks, they import information about other servers from the LAN and cell profiles. Servers, however, also export bindings to their own CDS namespace entries and export their names to the LAN and cell profiles. See the following sections on the server subtypes for further information on how servers are configured and located.

External time providers can be connected to servers, which propagate the precise time intervals they obtain from the time providers throughout the network.

Before one server can obtain time values from another, the servers must have the same epoch number. Epochs divide the DTS implementation into logically separate areas; servers only synchronize with other

servers that have the same epoch number. All servers have the same epoch number when they are created. Infrequently, you may need to change a server's epoch number using the management interface to isolate it from the network in order to correct a problem.

The Local Server Set

Local servers reside on the same LAN and maintain their clocks by synchronizing with each other. Because of the high throughput on this type of network, the skews between the local servers on a LAN are normally maintained at under 200 milliseconds. If at least one of the servers in the local set synchronizes with an accurate time provider, inaccuracies at each server may be less.

When a server is first initialized, it exports its binding to its entry in the namespace and adds its name entry to the LAN profile. Every server is automatically entered in the LAN profile for the related portion of the network. Local servers also import bindings from the LAN profile to build lists of servers with which they can synchronize.

Local servers perform time interval computations, adjust their clocks, and provide time values to each other for synchronization purposes. Each server attempts to synchronize with other servers (in a random fashion) in the local set at periodic intervals. At longer intervals, clerks request time values from the local servers. Clerks, however, need only to request intervals from the number of servers determined by the **minservers** attribute, which is usually a subset of all the local servers.

The Global Server Set

Local servers are available only to the servers and clerks that are in a single LAN, but global servers are available throughout a cell. Any server can be configured as either a local or a global server (see the DCE control program's **dts configure** command in the *DCE for AIX Administration Command Reference*).

The number of global servers is usually small, but global servers have several important functions that enable DTS to synchronize every node in the network. Global servers are necessary when:

- A network has multiple LANs or an extended LAN,
- Systems that are not on LANs have access to LANs through point-to-point links, and
- Clerks or local servers cannot access the required number of local servers determined by the **minservers** attribute.

You can configure a server as a global server by using the *dcecp* program's **dts configure** command with the **–global** option. Configuring a server as a global server causes the server to export its binding to its entry in the namespace and its name to the cell profile.

Local servers and clerks request time values from global servers when they cannot obtain the number of local server responses that are mandated by the **minservers** attribute. Certain local servers also regularly request the time from global servers.

Couriers

Local servers called *couriers* request time values from at least one randomly selected global server at every synchronization. When DTS starts up, it automatically sets the server's **courierrole** attribute value to **backup**. You can change the server's courier role by manually changing this attribute value. To do this, use the *dcecp* program's **dts modify** command with the **–change** option. If a server is connected to an external time provider, you should reconfigure it as a courier.

Couriers maintain lists of global servers whose bindings they import from the cell profile. At every synchronization, couriers use the responses of all local servers and one global server when synchronizing their own clocks. Couriers provide network-wide synchronization through the following procedure:

- Couriers request time values from at least one global server in a remote area and request the balance of values from local servers up to the number determined by the **minservers** attribute.
- Couriers use the global server times and local server times to synchronize the clocks that are in their respective systems.
- Couriers relay newly computed clock times to other servers and clerks on the LAN during future synchronizations.

For a network containing multiple LANs or point-to-point links, one server on each LAN or segment needs to be configured as a courier. This configuration ensures that various portions of the network remain synchronized and are not isolated from each other.

Using the management interface, you can also designate one or more servers to be backup couriers. These local servers temporarily assume courier functions in the event that no courier servers are available on the LAN. In such a case, the backup courier with the lowest ordered UUID regularly synchronizes with global servers until a courier is again available.

If a courier cannot find any global server, then it uses local servers and increments its **no global server detected** count.

DTS Planning for External Time Providers

Introduction — DTS Planning for External Time Providers

To closely synchronize your systems with UTC, you can place one or more time providers in your network. Time providers have many forms; they can be radio receivers, software/modem combinations, or satellite receivers. See *DCE for AIX Application Development Guide — Core Components* for additional information about the Time Provider Interface that you can use to integrate these devices in your network. See the “Time Providers and Time Services” on page 1-60, for a list of time sources.

More Detailed Information — DTS Planning for External Time Providers

If you plan to use time providers in your network, you can use one of the sample time provider programs that are supplied with the DTS software in **/opt/dcelocal/usr/examples/dts**. If you plan to use a time provider that does not have a sample program available, or you have special requirements, you can write a time provider program to match the time provider interface. After you select your time provider device and program, plan where to install the device in your network.

It is relatively simple to locate time providers to your best advantage. To do so, observe the following guidelines:

- Always locate a time provider at a server; if possible, locate the time provider at a server that is routinely accessed by the majority of servers in your network.
- Regardless of your network configuration, place the time providers where they will have the highest availability and use.
- If you have several segments to your network, and if you are using global servers to maintain synchronization across the network, locate the time providers on the global server systems.

Note: You cannot configure a server connected to a time provider as a courier. A server connected to a time provider never assumes the courier role, because the server process only solicits time values from the time provider. For additional information about courier servers, see “DTS Operations” on page 6-9.

DTS Timestamp Format

Introduction — DTS Timestamp Format

All responses to **dcecp** and **dtscp** program commands contain a timestamp that conforms to the input and output format shown in Figure 6-4.

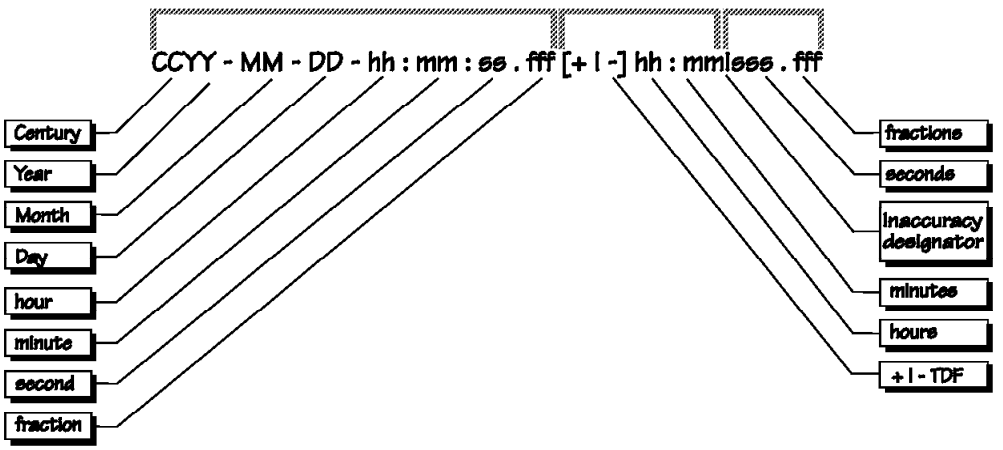


Figure 6-4. DTS Timestamp Syntax

More Detailed Information — DTS Timestamp Format

The timestamp uses the DTS format that is explained in "DTS Usage" on page 1-36. In this example, the year is 1994, the day is March 16, and the time is 14 hours, 29 minutes, and 47.52 seconds. A negative Time Differential Factor (TDF) of 5 hours and an inaccuracy of 3 milliseconds are included in the timestamp.

Example — DTS Timestamp Format

The following example shows a typical DTS time display:

```
1994-03-16-14:29:47.52000-05:00I000.003
```

General DTS Planning Guidelines

Consider the following questions as you plan your DTS implementation:

- Is your cell a single LAN, an extended LAN, a WAN, or a combination of LANs and WANs?
- What is the current or proposed network topology (component placement)?
- How many servers will be required? Where will they be located?
- Will global servers be required? Where will they be located?
- Will you need to configure any couriers if you are using global servers?
- Will you use an external time provider to obtain Coordinated Universal Time (UTC)?

Although there are many network configurations that affect DTS planning, several general rules apply regardless of your network configuration or the number of nodes in the network. These guidelines are summarized as follows:

- DTS must be installed with the other DCE components.
- Locate DTS servers on the same nodes as the servers for the other DCE components wherever possible.
- Each cell should have a minimum of three DTS servers, preferably four servers to provide redundancy.
- Each LAN should have at least one server.
- Locate the servers at the sites with the greatest number of nodes.

Although other factors must be considered when you plan your network, these factors are dependent on network topology and configuration.

Getting the Time from NTP Time Sources

Introduction — Getting the Time from NTP Time Sources

DTS provides two sample time provider programs:

dts_ntp_provider.c Takes the time from an NTP server as it would from a radio receiver. The user specifies the name of the NTP server and the inaccuracy.

dts_null_provider.c Is used on a DTS server whose clock is already synchronized by an external agent, such as NTP. It sets the inaccuracy, but prevents DTS from setting the time. The user sets the inaccuracy based on local experience with NTP. The null provider may be useful for sites that already have a radio clock that is managed by NTP. Make the node with the radio clock a DTS server and use the null time provider.

More Detailed Information — Getting the Time from NTP Time Sources

Getting the Time from Local NTP Time Sources

Run the DTS server on a node that is running an NTP clock driver with a clock and the null time provider. Specify the inaccuracy in a manner that is consistent with the time source (for example, a radio clock). Other DTS servers will take the time from this source. In this case, since the system is connected to a time source, it is an NTP Stratum 1 server.

Observe the following rules and advisories:

Rule:

- If this is the only local time source (radio clock) in the subnetwork, ensure that no other DTS node gives the time to NTP. If, however, there are other local time sources, this restriction does not apply.
- Do not run the null time provider if there is no local time source.

Advisory:

- Use a very small poll rate (about 1 second).
- Since NTP makes the **adjtime()** system call, occasionally the local node will have an unspecified inaccuracy.

Figure 6-5 on page 6-14 shows how a DTS server/client with a local time source takes time from an NTP Stratum 1 server.

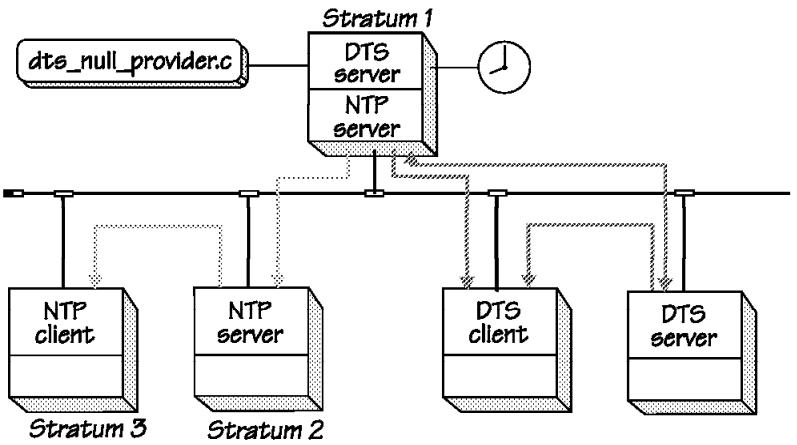


Figure 6-5. Local Time Source

Getting the Time from Remote NTP Time Sources

Run the DTS server with the NTP time provider (**dts_ntp_provider.c**) on a node with access to an NTP server. Specify the inaccuracy in a manner that is consistent with local NTP experience.

Observe the following advisories:

Advisory:

- If links to remote sources are distant, you may want one of the subnetwork nodes to run NTP locally.
- Remember that the NTP time provider does not accept time from an NTP node at Stratum 8 or higher.
- The NTP node needs to be as close to Stratum 1 as possible.

Figure 6-6 and Figure 6-7 both show a DTS server getting the time from a remote NTP time source that is a Stratum 3 server. However, in Figure 6-6 on page 6-15 (Scenario 1), all of the advisories made in the section are followed; in Figure 6-7 on page 6-15 (Scenario 2), the first advisory (running NTP locally on one of the subnetwork nodes if the link to a remote source is distant) is ignored.

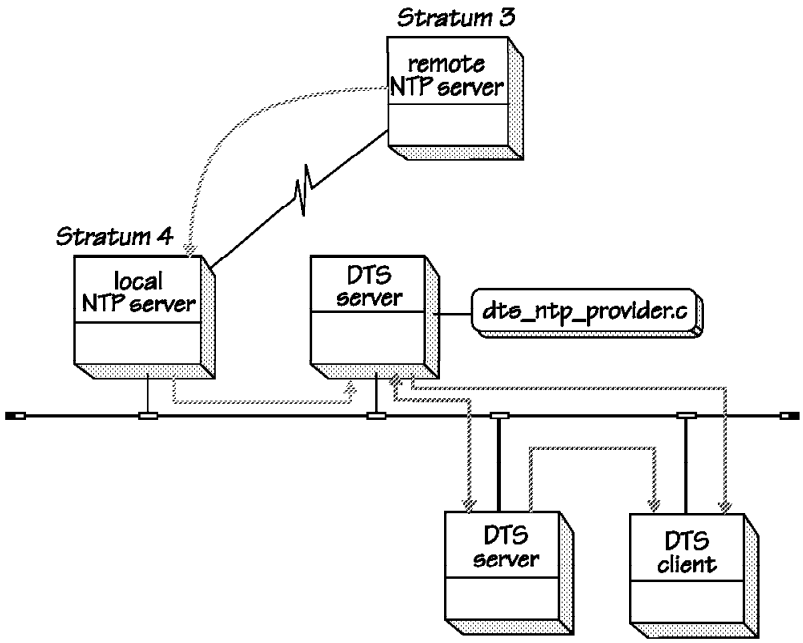


Figure 6-6. Getting the Time from a Remote NTP Time Source (Scenario 1)

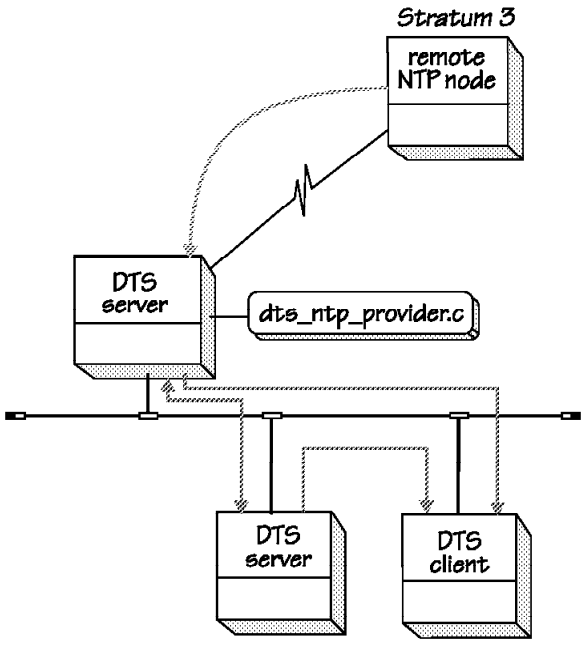


Figure 6-7. Getting the Time from a Remote NTP Time Source (Scenario 2)

Giving the Time to NTP Nodes

Introduction — Giving the Time to NTP Nodes

Any DTS server or clerk that runs the **ntpd** daemon or the **xntpd** daemon with the **–s** option and a special configuration file (**ntp.conf**) can be configured as an NTP server.

More Detailed Information — Giving the Time to NTP Nodes

For systems running the **ntpd** daemon, the **ntp.conf** configuration file must contain the following line:

```
peer /dev/null DTSS 8 -5 local
```

In addition, add **–s** to the **ntpd** entry in the /**sbin**/**init.d**/**ntpd** file or, for systems with **rc.local**, modify the line that starts **ntpd** accordingly.

For systems running the **xntpd** daemon, the **ntp.conf** configuration file must contain the following line:

```
peer 127.127.1.8
```

In addition, add **–s** to the **xntpd** entry in the /**sbin**/**init.d**/**xntpd** file or, for systems with **rc.local**, modify the line that starts **xntpd** accordingly.

In this configuration, NTP *never* sets the clock. NTP can, however, give the time to other NTP clients. Do *not* allow loops between DTS and NTP to form. If NTP gives the time to DTS and then DTS gives the time back to the same set of NTP servers, unexpected results can occur.

The NTP configuration file is set up to ensure that an NTP server that obtains the time from DTS is a Stratum 8 node. In addition, the **dts_ntp_provider** is prohibited from accepting time from a node at Stratum 8 or higher.

A DTS (server) node can give time to an NTP node if the following rules and advisories are observed:

Rule: The **ntp.conf** file must declare this node at Stratum 8.

Advisory:

- Multiple nodes in the set can be running **ntpd –s** or **xntpd –s**.
- If any DTS-managed system has a local time source, that system should be used as an NTP **–s** server.
- Although this operation can occur on either a DTS server or a DTS client node, a DTS server is preferred.

Note: If null providers are used, the rules in “Getting the Time from NTP Time Sources” on page 6-13 must also be followed because null providers running on NTP nodes can bypass the stratum check.

Figure 6-8 on page 6-17 shows two DTS server nodes running **ntpd –s** and providing time to an NTP subnetwork. The **ntp.conf** file defines these servers at Stratum 8.

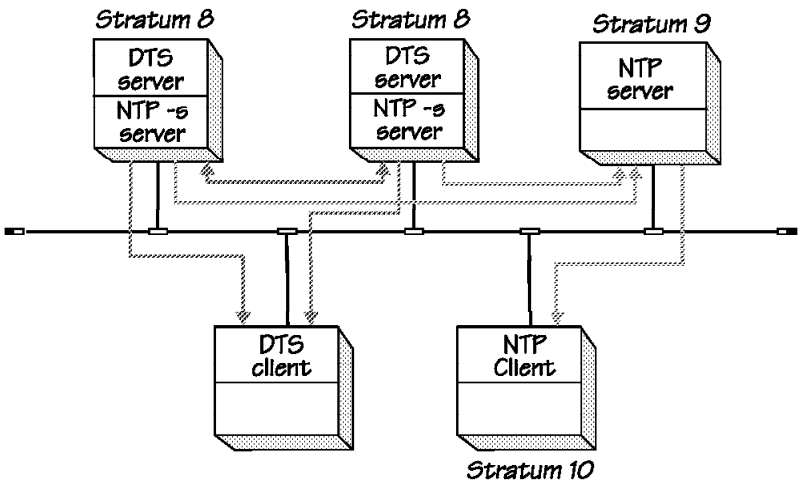


Figure 6-8. Giving the Time to NTP

Managing DTS with dcecp Commands

Introduction — Managing DTS with dcecp Commands

"**dcecp** Commands Usage" on page 2-19 introduces the **dcecp** program and its command syntax, so this section does not repeat this information. It simply describes some **dcecp** commands available for managing DTS.

More Detailed Information — Managing DTS with dcecp Commands

DTS Objects

This section describes the commands that the **dcecp** program provides specifically for managing DTS.

The **dcecp** program's commands for DTS perform various operations on objects representing components of the service. For example, the **dts stop** command stops the server or clerk on the local node. The following sections describe the DTS objects upon which the **dcecp** program operates and the types of operations that the control program can perform on these objects.

The **dcecp** program has functions that operate on the following DTS objects:

dts This object represents either:

- A local or global server that supplies the time to client applications and systems in a distributed computing environment, or
- An intermediary program that plays the role of a clerk on a client system. DTS clerks obtain the time from a DTS server and adjust the clock.

clock This object represents the local system's clock and the time that the clock tells.

DTS Command Operations

The following list summarizes the operations performed by the **dcecp** program object commands on DTS objects.

Command Operations	Descriptions
activate	Changes the state of the clerk or server process from inactive to active and causes the object to synchronize its time.
configure	Configures a server as a global or local server.
deactivate	Changes the state of a clerk or server process from active to inactive and causes the object to stop synchronizing its time.
help	Displays a list of operations that can be performed on the clerk, server, or clock, or a verbose description of the specified object.
modify	Modifies the attribute information for a clerk or server.
operations	Displays a short list of the operations that can be performed on the clerk, server, or clock.
set	Sets the clock gradually or immediately to the time specified by the argument (in DTS-style timestamp format).
show	For a clerk or server, displays information about attributes or counters. For a clock, displays the clock's time in the DTS-style timestamp format.
stop	Stops the clerk or server process.
synchronize	Tells the **dtsd** to gradually or immediately synchronize (the **–abruptly** option) with the DTS servers.

DTS Object Attributes and Counters

DTS clerk and server objects have attributes and counters which are pieces or sets of data that reflect or affect their operational behavior. Some DTS clerk and server attributes are used internally by the DTS daemon and you are only allowed to view the values (by using the **dcecp** program's **dts show** command). Other attributes contain values that you can reset according to the needs of your environment (by using the **dcecp** program's **dts modify** command). Counters are used internally by the DTS daemon and contain values that you can only view.

The following lists the server and clerk attributes you can set.

Servers	Clerks
checkinterval	—
courierrole	—
epoch	—
globaltimeout	**globaltimeout**
localtimeout	**localtimeout**
maxinaccuracy	**maxinaccuracy**
minservers	**minservers**
queryattempts	**queryattempts**
serverentry	—
servergroup	—
serverprincipal	—
syncinterval	**syncinterval**
tolerance	**tolerance**

The following lists the server and clerk attributes you cannot set.

Servers	Clerks
actcourierrole	—
autotdfchange	**autotdfchange**
clockadjrate	**clockadjrate**
clockresolution	**clockresolution**
globalservers	**globalservers**
lastsync	—
localservers	**localservers**
maxdriftrate	**maxdriftrate**
nexttdfchange	**nexttdfchange**
provider	—
status	—
tdf	**tdf**
timerep	**timerep**
type	**type**
uuid	**uuid**
version	**version**

For detailed descriptions of both the DTS server and clerk attributes and counters, see the **dts** object command's reference page in the *DCE for AIX Administration Command Reference*.

Management Tasks Specific to Servers

Introduction — Management Tasks Specific to Servers

Managing DTS servers involves some special tasks. These additional tasks include:

- Setting server epoch numbers
- Assigning the courier role to a server
- Designating a server as a global server
- Setting the attributes for a connection to a time provider

More Detailed Information — Management Tasks Specific to Servers

Designing Global and Courier Servers

If your network has WAN links or is an extended LAN, you may need to use global and courier servers to synchronize the nodes in separate network segments. To synchronize nodes across a network, you assign global roles to some servers and courier roles to selected local servers. See “General DTS Planning Guidelines” on page 6-12 for advice on planning the location of global and local courier servers. To assign server roles, follow the instructions in the following sections.

Advertising Global Servers

To assign a server to the global set of servers, you must advertise the server with the **dcecp** program's **dts configure** command. Advertising the server simultaneously adds binding information to the server's CDS name and also adds the server's entry to the cell profile. Since CDS and the cell profile are available to every node in your network, DTS can perform a lookup in the cell profile to obtain the locations of nodes that it cannot reach on the LAN.

Assigning the Courier Role to Servers

Courier servers play an important role in maintaining synchronization between the systems in separate parts of your network. A courier server requests a time value from at least one global server at every synchronization. This procedure enables a courier server to propagate times from remote systems to a LAN or local area, thereby keeping the LAN in synchronization with all the other parts of the network.

There are three courier roles that you can assign to a server (the **courierrole** attribute) — **backup**, **courier**, and **noncourier**. The default courier role for a global or local server at its creation is **backup**.

Use the **courier** setting for the **courierrole** attribute to designate a server as the primary link to other portions of your network. Use the **backup** setting to designate a server as a secondary link to other areas of the network. A backup courier is only effective if no other courier is available on the LAN.

Note: There are no significant processing or overhead penalties associated with the backup courier role; you can designate one of the servers on a LAN as a courier and designate all the other servers on the LAN as backup couriers. If you have configured several servers as backup couriers and the courier becomes unavailable, the backup courier with the lowest ordered UUID becomes the effective courier.

Matching Server Epochs

At startup, a server's epoch number must match those of the other servers with which it synchronizes. When synchronizing, a server disregards clock values that are from servers whose epoch numbers do not match its own.

When DTS servers are initially enabled, the epoch number for each servers is 0, so you need not change the epoch numbers at initial installation. Later, if you add a server to an existing network or change a clerk to a server, ensure that the new server and the pre-existing servers have matching epoch numbers.

Examine the attributes list that the command returns for the server's **epoch** attribute value. If the epoch of a server that you just created matches those of the other servers, the new server can synchronize immediately. If the epochs do not match, however, and you do not change the epoch of the new server, the new server ignores the pre-existing servers.

When you know that a server is starting up with the proper epoch number, *do not* change the epoch unless serious system or network problems corrupt all of the server clock values. In the unlikely event that the majority of the server clocks become faulty, use the **dts set** command and the **clock set** command to isolate problem servers so that you can perform troubleshooting and maintenance without affecting the rest of the DTS application.

Setting the checkinterval Attribute for Connection to a Time Provider

Set the **checkinterval** attribute only for servers that are connected to time providers. DTS uses the **checkinterval** attribute to periodically check all the servers on a LAN to make sure that they remain synchronized with the time provider. When the amount of time specified by the **checkinterval** attribute setting has elapsed, the server with the time provider (the TP server) performs the following procedure:

1. The TP server requests time values from all the other servers on the LAN.
2. The TP server starts the synchronization process.

3. The TP server identifies the server time intervals that do not intersect with its own.
4. The TP server issues event messages for each faulty server it detects.

In the previous sequence, the TP server does not actually set the system clock after it starts the synchronization process. The TP server merely runs the process to detect faulty servers. The DTS software assumes that the time value at the TP server is the most accurate available, so the TP server does not use the values it collects from other servers to change its clock. Instead, the TP server functions as a reference timekeeper for the other servers.

You can set the check interval to a lower value for a more rapid notification of faulty servers, but be aware that lower settings can increase the load on network resources.

Examples — Management Tasks Specific to Servers

- The following example shows how to advertise a server as a global server, thereby registering it with CDS and entering it in the cell profile:

  ```
  dcecp> dts configure -global
  ```

  The **–global** option designates that a server should be configured as a global server, rather than a local server.

- To remove a server's designation as a global server, use the **dts configure** command as follows:

  ```
  dcecp> dts configure -notglobal
  ```

  This command unadvertises the global server, removing its entry from the cell profile and its binding information from its CDS name.

- To assign the courier role to a server, enter the following **dcecp** program command:

  ```
  dcecp> dts modify -change {courierrole courier}
  ```

- To assign the backup courier role to a server, enter the following:

  ```
  dcecp> dts modify -change {courierrole backup}
  ```

- Enter the DCE control program's **dts show** command to find out the epoch number of a server:

  ```
  dcecp> dts show /.:/hosts/orion/dts-server
  ```

- The following example shows how to change a server's epoch number after you enable the server:

  ```
  dcecp> clock set -abruptly -epoch 0
  ```

- The following example shows how to set the **checkinterval** attribute value:

  ```
  dcecp> dts modify /.:/hosts -change {checkinterval 00-00:00:30.0000}
  ```

Modifying Clerk and Server Attributes

Introduction — Modifying Clerk and Server Attributes

Many management tasks involve modifying the attributes of DTS clerks and DTS servers. The **dcecp** program has several commands for displaying and changing the attributes of these entities.

More Detailed Information — Modifying Clerk and Server Attributes

To display the attribute values of a DTS clerk or DTS server, use the **dts show** command. (You can also use the **dts show** command to view the values of DTS entity counters; however, you cannot modify counter values.)

To modify the attributes for a DTS clerk or server, use the **dcecp** program's **dts modify** command. Several examples of this command appear in the following sections. These sections include descriptions of the settable attributes for clerks and servers. These sections also offer suggestions for various attribute settings, depending on your network configuration.

The minservers *Characteristic*

The **minservers** attribute specifies how many servers must supply time values to the system before DTS can synchronize the system clock.

The default and minimum recommended value for the **minservers** attribute is **3**. Your system requires values from three servers in order to compute a reliable new time. Depending on whether it is a server or clerk, the system has different requirements of the other systems in the network:

- A clerk requires values from three servers.
- A server requires values from two other servers. Each server uses its own clock value when computing a new time.

To reset the **minservers** attribute value, enter the **dts modify** command with the **–change** option set to the desired value. The command accepts values from **1** to **10**.

Although there is no direct relationship between the **localservers** attribute (which specifies the number of local servers in a LAN) and the **minservers** attribute, the **minservers** attribute value is usually a subset of all the local servers. To see the current values of either or both of these attributes, use the **dts show** command. Wait until the DTS nodes on your LAN are running for at least 10 minutes before issuing the command. That way, the **dts show** command is sure to show all of the local servers in your node's synchronization list.

Whenever the system cannot contact the number of servers specified by the **minservers** attribute setting, the system increments the **toofewservers** counter, logs the event, and displays the `Too Few Servers Detected` event message. Information included in the even message shows the number of servers that are currently available and the number required. If you see this event message displayed, check whether any of the servers have failed, test the communications links to ensure that the system has not been isolated from the servers, or add servers to the network.

You can use the **minservers** attribute in other ways, depending on your network configuration. Consider the following cases:

- If you have only a few systems in your network and you want to synchronize the nodes regardless of server drift, lower the **minservers** attribute value to **1** or **2**. Although the resulting synchronized time is a less reliable measure of UTC, you increase the likelihood that the systems will synchronize. If the setting is less than 3, however, the system cannot identify faulty servers. Subsequent server clock drift causes divergence from UTC.
- To increase fault tolerance and ensure that the systems compute reliable times, set the **minservers** attribute value to **3** (the default setting) or higher. The systems can then identify faulty servers and compute the narrowest overlapping interval for the time values that they receive. Remember, however, that your system will not synchronize until there are at least three servers available.

The number of nodes in your network and the types of applications that you use determine whether guaranteed synchronization or reliable times and fault tolerance are more important.

Use of **minservers** *with Global Servers*

If your network consists of more than a single LAN, it should have a set of global servers. You can create global servers by advertising local servers to the cell profile.

The presence of global servers in your network can influence the value that you choose for the **servers required** characteristic. If the number of local servers available to a clerk or server is less than the **minservers** attribute setting, the clerk or server automatically searches the cell profile for a global server name. The clerk or server then requests time values from the global and local servers.

You can check to see whether global servers exist by entering the **dts show** command and viewing the **globalservers** attribute value.

Use of the **minservers** *Attribute with Systems on Point-to-Point Lines*

If you are using DTS on a system that connects to a LAN through a point-to-point WAN link, the solitary system never has more than one local server available. The recommended **minservers** attribute setting for such a system is **e**. If the system is configured as a clerk, it does not have any local servers and must query three global servers to synchronize. If the system is configured as a server, it must query two global servers to synchronize.

The **maxinaccuracy** *Attribute*

The **maxinaccuracy** attribute specifies the greatest allowed bound on your system's inaccuracy before DTS causes the system to synchronize. When the system exceeds the bound determined by the **maxinaccuracy** attribute setting, DTS forces the system to synchronize until the inaccuracy is reduced to a level that is at or below the setting. Use the **maxinaccuracy** attribute setting as a trigger for synchronization. You can change the setting to vary the tolerance of intersystem synchronizations, but be aware that as the setting becomes lower, network overhead rises. The default setting is 0.10 seconds (100 milliseconds).

The effects of the **maxinaccuracy** attribute setting on the system's synchronization behavior are as follows:

1. The system's clock value accumulates more inaccuracy than the **maxinaccuracy** attribute value and DTS initiates a synchronization.
2. DTS computes a new time value.
3. DTS adjusts the system clock.
4. If the new clock setting still exceeds the **maxinaccuracy** attribute value, or if clock drift later causes the inaccuracy to reach the value, the cycle is repeated.

Note: If synchronization repeatedly fails to achieve an inaccuracy that is less than the **maxinaccuracy** attribute value, the system can be continuously synchronizing. See “The **syncinterval** Attribute” in this section for information on how the **syncinterval** attribute prevents this loop.

The default **maxinaccuracy** attribute value is designed to keep the system accurate enough for most applications without being intrusive to network communications or system processing. If your network includes one or more time providers that ensure extremely low inaccuracy, you can lower the **maxinaccuracy** attribute value. Raise the value if:

- A time provider is not used in the network,
- A system is part of a WAN-only network configuration, or

- The applications that call DTS do not require the level of precision achieved by the default setting.

The syncinterval *Attribute*

The **syncinterval** attribute prevents your system from synchronizing more often than the specified interval. This characteristic prevents the **maxinaccuracy** attribute from causing continuous synchronizations. As mentioned in "The **maxinaccuracy** Attribute" in this section, the **maxinaccuracy** attribute triggers system synchronization as long as the system's inaccuracy is above a specified value. The **syncinterval** attribute prevents synchronization from occurring more frequently than the hold-down value. (The value is randomized to prevent several systems from synchronizing simultaneously and is an average rather than an exact value.)

The **maxinaccuracy** and **syncinterval** attributes are interdependent; system synchronization occurs automatically when *both* of the following conditions are met:

- The inaccuracy of its clock equals or exceeds the **maxinaccuracy** attribute value.
- The time since the last synchronization equals or exceeds the **syncinterval** attribute value (slightly randomized).

Note: If the system reaches the **syncinterval** setting, but has not yet reached the **maxinaccuracy** setting, the system does not synchronize.

The default **syncinterval** attribute value for servers and for clerks is 24 hours. If you are trying to minimize the skew between systems, you can lower the **syncinterval** attribute value.

The **syncinterval** attribute does not prevent the **clock synchronize** command from working. You can synchronize a system at any time by entering this command. The **syncinterval** attribute affects only automatic synchronizations triggered by the **maxinaccuracy** attribute. See the *DCE for AIX Administration Command Reference* for more information on the **clock synchronize** command.

The tolerance *Attribute*

The **tolerance** attribute determines how DTS reacts if the system clock becomes faulty. A faulty clock is a rare condition, but some causes of faulty clocks include:

- Defects in the clock hardware, including clock drift that is greater than the manufacturer's specifications;
- Malfunctioning time providers;
- Hardware clock ticks that are lost by the operating system; and
- The system memory containing the clock value is corrupted.

During the synchronization process, DTS detects that a system's clock is faulty if the clock value and its inaccuracy fail to intersect with those of the servers used for synchronization. This process is shown in Figure 6-9 on page 6-25, where value **t2** is faulty.

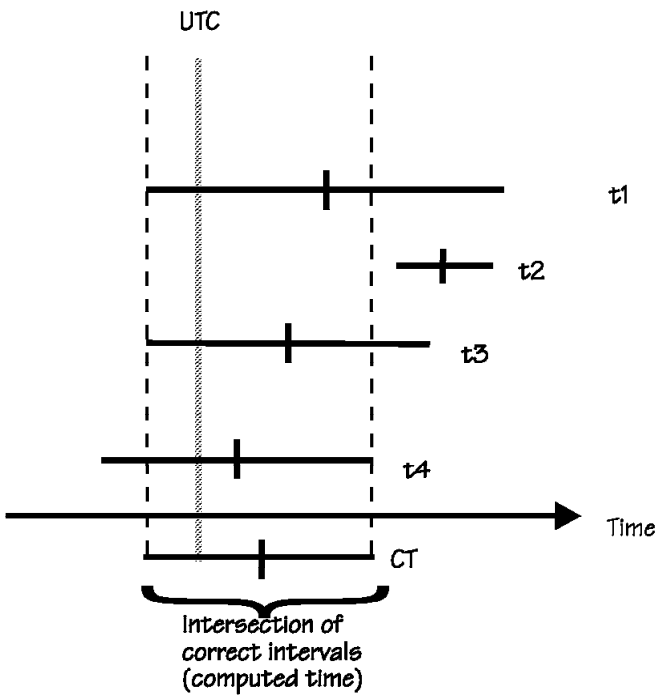


Figure 6-9. Local Fault

If DTS detects a faulty system clock during synchronization, the severity of the fault and the system's **tolerance** attribute setting determine how DTS reacts. When the fault is detected, DTS performs one of the following operations:

- If the faulty time interval that is supplied by the clock is within the bounds of the error tolerance, DTS increases the inaccuracy of the value supplied by the clock and adjusts the clock gradually.
- If the faulty time interval that is supplied by the clock is outside the bounds of the error tolerance, DTS immediately sets the clock to the new computed time.

Before you change the default **tolerance** setting (5 minutes), determine the requirements of the applications that use the system time. Some distributed applications, such as a CDS server, require that systems have no more than 5 minutes of inaccuracy. Larger error tolerances may prevent such applications from properly sequencing CDS namespace entries. For these applications, set the **tolerance** attribute value to 5 minutes or less.

Some applications may require DTS to adjust the system clock gradually and monotonically (forward). You can increase the **tolerance** attribute setting for these applications to ensure that the clock is abruptly set only in the event of a catastrophic error. If you could set the **tolerance** attribute to infinity, you could guarantee that the clock is never set abruptly. This setting is not available, but you can enter any setting less than 10675199-00:00:00.000 (approximately 29,227.5 years).

The localtimeout*,* globaltimeout*, and* queryattempts *Attributes*

When a system queries a server, it waits for a response for the period that is specified by the **localtimeout** or **globaltimeout** attribute. The **localtimeout** attribute setting applies when the system attempts to contact a local server; the **globaltimeout** attribute setting applies when the system attempts to contact a global server.

The **queryattempts** attribute determines how many times DTS resets the time-out timer before the system quits trying to contact a given server. When the time-out setting exceeds the number of times specified by the **queryattempts** attribute, the system stops querying the server. If the system is querying a global

server, DTS then generates a `Server Not Responding` event report and removes the server from the system's list of global servers. If a response from the global or local server is required in order to meet the **minservers** attribute setting, DTS generates a `Too Few Servers` event report and the system does not synchronize.

The default setting for the **queryattempts** attribute is **3**.

The default setting for the **localtimeout** attribute is 2 seconds and the default setting for the **globaltimeout** attribute is 15 seconds. The global setting is larger to account for the communications delay on WAN links that are often used to access the global set. It is unlikely that you will have to change the **localtimeout** attribute setting. The **globaltimeout** attribute setting, however, may need to be changed due to the variations in WAN topologies and transmission quality.

If you continually receive `Server Not Responding` event reports for a global server, increase the **globaltimeout** attribute setting. If you increase the setting and the event reports continue, there may be a problem with the communications link to the server.

The serverentry and serverprincipal Attributes

During the initial configuration of DCE and DTS, one DTS entry name is created for use with CDS and one DTS name is created for use with the Service. If you subsequently want to change the name of a server, you can do this by changing two of the server's attributes — **serverentry** and **serverprincipal**. The default settings for those **dcecp** program attributes are the same as the default settings for the names that are created during the initial DCE configuration; they are the recommended settings. This section describes additional considerations for the settings of these attributes. If you decide to change the settings of the **serverentry** and **serverprincipal** attribute values, be sure that the new values are appropriate. If not, you will experience trouble with DTS.

The **serverentry** attribute specifies the CDS entry name where bindings for the server are exported. If you change the setting of this attribute, the entry is also modified in the namespace.

The **serverprincipal** attribute specifies the principal name of the server which is used for authentication. If you change the name using the **dcecp** program, you must create a matching principal name and account in the Security Service registry. When you do this, you must add the new principal name to the existing DTS server group (**dts-servers**). The machine principal must be a member of this authorization group. See “Creating and Maintaining Accounts” on page 1-48 for further information on creating a new principal account and see “Creating and Maintaining Principals, Groups, and Organizations” on page 1-48 for information on adding a principal name to an existing server group.

Examples — Modifying Clerk and Server Attributes

- To display the attribute values for all the clients and servers on the local node, enter the following command:

```
dcecp> dts show
{checkinterval +0-01:30:00.000I-----}
{epoch 0}
{tolerance +0-00:10:00.000I-----}
{tdf -0-05:00:00.000I-----}
{maxinaccuracy +0-00:00:00.100I-----}
{minservers 3}
{queryattempts 3}
{localtimeout +0-00:00:05.000I-----}
{globaltimeout +0-00:00:15.000I-----}
{syncinterval +0-00:02:00.000I-----}
{type server}
{courierrole backup}
{actcourierrole courier}
```

```
{clockadjrate 10000000 nsec/sec}
{maxdriftrate 1000000 nsec/sec}
{clockresolution 10000000 nsec}
{version V1.0.1}
{timerep V1.0.0}
{provider no}
{autotdfchange no}
{nexttdfchange 1994-10-30-06:00:00.000+00:00I0.000}
{serverprincipal hosts/gumby/self}
{serverentry hosts/gumby/dts-entity}
{servergroup subsys/dce/dts-servers}
{status enabled}
{uuid 000013ed-000b-0000-b8ef-03a4fcdf00a4}
```

This example shows the attribute values for the single server located on the local node. The attributes that the **dts show** command displays for a clerk are different. Also, more attributes are displayed for a server (see "More Detailed Information — Managing CDS with **dcecp** Commands" on page 5-42 and "More Detailed Information — Managing DTS with **dcecp** Commands" on page 6-17).

- To increase the required number of servers to four, issue the following command:

```
dcecp> dts modify -change {minservers 4}
```

- The **dts show** command can be entered with options (**–attributes** or **–all**) or without any options, as follows:

```
dcecp> dts show
{checkinterval +0-01:30:00.000I-----}
{epoch 0}
{tolerance +0-00:10:00.000I-----}
{tdf -0-05:00:00.000I-----}
{maxinaccuracy +0-00:00:00.100I-----}
{minservers 4}
{queryattempts 3}
{localtimeout +0-00:00:05.000I-----}
{globaltimeout +0-00:00:15.000I-----}
{syncinterval +0-00:02:00.000I-----}
{type server}
{courierrole backup}
{actcourierrole courier}
{clockadjrate 10000000 nsec/sec}
{maxdriftrate 1000000 nsec/sec}
{clockresolution 10000000 nsec}
{version V1.0.1}
{timerep V1.0.0}
{provider no}
{autotdfchange no}
{nexttdfchange 1994-10-30-06:00:00.000+00:00I0.000}
{serverprincipal hosts/gumby/self}
{serverentry hosts/gumby/dts-entity}
{servergroup subsys/dce/dts-servers}
{status enabled}
{uuid 000013ed-000b-0000-b8ef-03a4fcdf00a4}
```

In the previous example, the **minservers** attribute value is set to **4**. This setting provides redundancy. In the case where there are no global servers in the network, the system synchronizes even if a local server becomes unavailable.

- The **dts show** command can be entered with options (**–attributers** or **–all**) or without any options, as follows:

```
dcecp> dts show
{checkinterval +0-01:30:00.000I-----}
{epoch 0}
{tolerance +0-00:10:00.000I-----}
{tdf -0-05:00:00.000I-----}
{maxinaccuracy +0-00:00:00.100I-----}
{minservers 3}
{queryattempts 3}
{localtimeout +0-00:00:05.000I-----}
{globaltimeout +0-00:00:15.000I-----}
{syncinterval +0-00:02:00.000I-----}
{type server}
{courierrole backup}
{actcourierrole courier}
{clockadjrate 10000000 nsec/sec}
{maxdriftrate 1000000 nsec/sec}
{clockresolution 10000000 nsec}
{version V1.0.1}
{timerep V1.0.0}
{provider no}
{autotdfchange no}
{nexttdfchange 1994-10-30-06:00:00.000+00:00I0.000}
{serverprincipal hosts/gumby/self}
{serverentry hosts/gumby/dts-entity}
{servergroup subsys/dce/dts-servers}
{status enabled}
{uuid 000013ed-000b-0000-b8ef-03a4fcdf00a4}
```

- The following example shows how to change the **maxinaccuracy** attribute value to 0.2 seconds:

  ```
  dcecp> dts modify -change {maxinaccuracy 00-00:00:00.200}
  ```

- If you want a clerk to synchronize every 5 minutes if its inaccuracy reaches 100 milliseconds, enter the following command:

  ```
  dcecp> dts modify -change {syncinterval 00-00:05:00.000}
  ```

- The following example shows how to change the **tolerance** attribute setting to 3 minutes:

  ```
  dcecp> dts modify -change {tolerance 00-00:03:00.000}
  ```

- The following example shows how to set the **queryattempts** attribute value:

  ```
  dcecp> dts modify -change {queryattempts 4}
  ```

- In the following example, the **globaltimeout** attribute setting is changed to 20 seconds:

  ```
  deccp> dts modify -change {globaltimeout 00-00:00:20.000}
  ```

- The following command sets the **serverentry** attribute value:

  ```
  dcecp> dts modify -change {serverentry /.:/hosts/cyclops/dts_ref_node}
  ```

- The following command sets the **serverprincipal** attribute:

  ```
  dcecp> dts modify -change {serverprincipal /.:/hosts/ajax/dts_machine}
  ```

Preventing Loops

Introduction — Preventing Loops

Do *not* allow loops (such as NTP → DTS → NTP) to form.

Run the null time provider (**dts_null_provider.c**) only if you have a local time source. If you do not have a local time source, you can run the null time provider, but do *not* disseminate NTP time anywhere in the local set.

More Detailed Information — Preventing Loops

Figure 6-10 on page 6-29 shows a configuration that is *not* recommended. This configuration only works as long as the remote NTP Stratum 2 node does not fail.

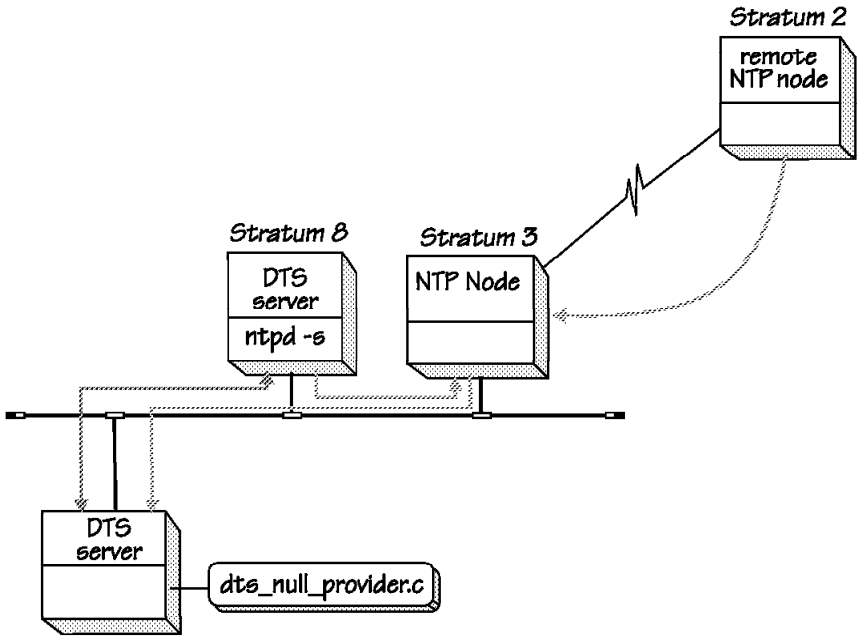


Figure 6-10. Configuration Before Stratum 2 Node Fails

If the remote NTP Stratum 2 node fails, the Stratum 3 node starts accepting time from the Stratum 8 node. When this occurs, the Stratum 3 node drops to Stratum 9 node drops to Stratum 10, as shown in Figure 6-11.

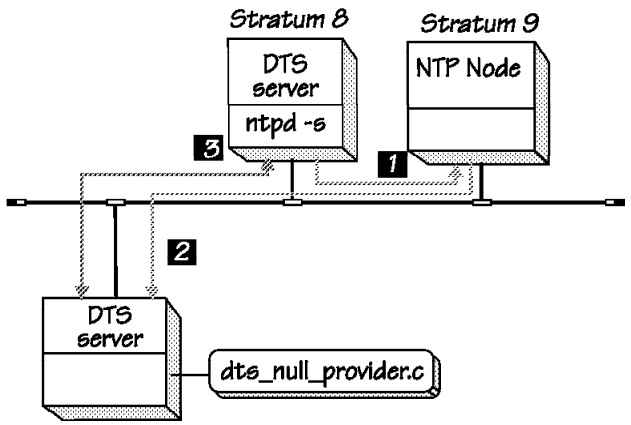


Figure 6-11. Configuration After Stratum 2 Node Fails

This scenario in Figure 6-11 shows the creation of a loop:

1. From the node that is labeled Stratum 8, proceed to the NTP node that is labeled Stratum 9.
2. From the NTP node that is labeled Stratum 9, continue to the node that is labeled Stratum 10.
3. DTS then feeds the time back to the node that is labeled Stratum 8, creating a loop.

If this occurs, time in the NTP and DTS subnetwork can drift from UTC.

Reconfiguring DTS on Nodes

Introduction — Reconfiguring DTS on Nodes

DTS is initially configured during the overall DCE configuration procedure for a node (see *DSS for AIX Up and Running!.* for configuration information). The DCE configuration procedure automatically creates and activates DTS servers and DTS clerks on designated nodes. You can, however, reconfigure DTS on a node at any time.

More Detailed Information — Reconfiguring DTS on Nodes

Stopping an Existing Clerk or Server

To stop the existing DTS clerk or DTS server on a node, use the **dcecp** program's **dts stop** command. Execution of this command first deactivates the clerk or server (that is, disables the function by which the clerk or server synchronizes the system clock), then stops the process. You enter the **dts stop** command as follows:

```
dcecp> dts stop
```

The **dts stop** command calls the **dcecp** program's **dts deactivate** command to deactivate the clerk or server process. This is the command that you should use whenever you want to deactivate a clerk or server process, but not stop it. You enter the **dts deactivate** command as follows:

```
dcecp> dts deactivate
```

Creating a New Clerk or Server

To create a new clerk or server on the node, use the configuration process that configures additional DTS clerks and servers (see *DSS for AIX Up and Running!*). The configuration process for configuring additional clerks and servers restarts the DCE daemon (**dtsd**) as either a clerk or a server.

As in the initial DTS configuration, if you are creating a server, you must tell the configuration process the type of server that it is to create — global or local. Before you choose the server type, consider the role that the server will play in propagating the network time.

Local servers can have a noncourier role (the value of the **courierrole** attribute is set to **noncourier**). A noncourier server does not participate in time propagation. Local servers can also have a courier role (the value of the **courierrole** attribute is set to **courier**) or a backup courier role (the value of the **courierrole** attribute is set to **backup**). Courier servers have primary responsibility for synchronizing the clocks between the nodes in a segment of the network. Backup couriers are secondary links, which propagate the time when no courier server is available. When you create a local server, the courier role is automatically set to **backup**.

Global servers must play the **noncourier** role. They cannot be designated as couriers or backup couriers.

Setting Clerk and Server Attribute Values

After you have created a new clerk or server on a node, you will want to set certain attribute values.

If you reconfigure a node to be a server, you need to match the epoch (the **epoch** attribute value) of the newly-created server to the epoch that is shared by the pre-existing servers in the network segment. You want to do this so that the new server can synchronize immediately with these servers. Instructions for changing server epoch numbers are given in “Matching Server Epochs” under “More Detailed Information — Management Tasks Specific to Servers” on page 6-19.

You may also want to check the rest of the attributes that apply only to servers to see that they complement the value settings of the attributes for pre-existing servers. For example, if the server has an external time provider, you may want to check the **checkinterval** attribute. This attribute specifies the amount of time that the server waits before synchronizing with the other servers on the LAN.

If you have changed your mind about a server's courier role since you created the server, you can modify the **courierrole** attribute value.

If you created a clerk, you may want to check the new clerk's attribute values against those of the pre-existing clerks and servers in the network.

General instructions for modifying the attributes of DTS clerks and DTS servers are covered in "Modifying Clerk and Server Attributes" on page 6-21.

Steps — Reconfiguring DTS on Nodes

If you choose to reconfigure DTS, you must perform the following steps:

1. Stop the clerk or server process (DTS daemon) that is currently running on the node.
2. Run the configuration process to restart the DTS daemon on the node as a clerk or server.
3. Set any clerk or server attribute values as needed.

Temporarily Reconfiguring DTS

Introduction — Temporarily Reconfiguring DTS

From time to time a situation or problem may arise in your network that requires you to temporarily reconfigure DTS on one or more nodes. Perhaps a node in the LAN is having problems and you need to have another node take over the clerk or server role of the problem node. Rather than adding an unnecessary server or clerk to the network, you can convert the clerk or server so that it plays the needed role.

If you convert a clerk or server, the change is only temporary. When DCE is stopped and restarted on a node, the node reverts to its initial DTS configuration. A node that was initially configured as a DTS server becomes a server; a node that was initially configured as a DTS clerk becomes a clerk. In order to permanently change the DTS configuration on a node, you must run the configuration process as discussed in *DSS for AIX Up and Running!*.

Steps — Temporarily Reconfiguring DTS

To temporarily convert a clerk to a server (or a server to a clerk), perform the following steps:

1. Stop the clerk or server process that is currently executing on the node by using the **dcecp** program's **dts stop** command:

   ```
   dcecp> dts stop
   ```

   After you stop the clerk or server, quit the **dcecp** program.

2. Restart the DTS daemon on the node as a clerk or server by executing the **dtsd** command with the appropriate option (the **–c** option for a clerk or the **–s** option for a server). For example, to create a local server, enter the following command:

   ```
   > dtsd -s
   ```

This command creates a local server that is a backup courier (the server's **courierrole** attribute value is set to **backup** by default). If desired, you can designate another courier role for the server in the **dtsd** command line by using the command's **–k** option. Other than a backup courier, the local server created in the example can be a courier (**courier**) or can have no courier role (**noncourier**).

In the following example, the local server is given the role of a courier:

```
> dtsd -s -k courier
```

To create a global server, enter the **dtsd** command with the **–g** option:

```
> dtsd -s -g
```

Note: If you are reconfiguring a node that previously ran a DTS clerk so that it runs a DTS server, you need to perform extra steps. You must create a principal account for the new server in the Security Service registry and you must add the server's name to the existing DTS server group (**dts-entity**). Otherwise, DTS clerks will not be able to find the newly-created server. For instructions on creating a principal account, see “Creating and Maintaining Accounts” on page 1-48. For instructions on adding a principal name to a group, see “Creating and Maintaining Principals, Groups, and Organizations” on page 1-48.

3. Set any clerk or server attribute values as needed by using the DCE control program's **dts modify** command.

Chapter 7. DCE Security Service

This section describes security-related tasks in the Distributed Computing Environment. For introductory and overview information on DCE security, see “DCE Security Overview” on page 1-44. That section also groups related tasks together in the order in which they may normally occur so you can access these referenced tasks in a logical, tutorial-type manner. You may find this helpful in performing related tasks together from beginning to end.

ACL Entries and Masks

Introduction — ACL Entries and Masks

ACL entries are of several different ACL entry types, each type being for a particular purpose. All ACL entries are represented in a uniform list syntax.

More Detailed Information — ACL Entries and Masks

ACL Syntax

The DCE control program uses the command syntax that is supported by the Tcl language. Within Tcl, the list that represents an ACL entry contains either two or three elements, depending on the ACL entry type, and is in the following form:

{*type***[***key***]** *permissions***}**

ACL Entry Types for Principals and Groups

ACL entry types let you define entries for:

- Principals and groups
  - Principals and groups in the local cell
  - Principals and groups in foreign cells
  - Delegate entries
  - All principals in the local cell for whom individual ACL entries have not been created
  - All principals in the local and all foreign cells with privilege attributes that do not match any of the other ACL entries
- Masks used for authenticated and unauthenticated users
- As-yet-undefined entry types that can be copied and displayed (if not interpreted) by dissimilar DCE releases

If any principal or group is not authenticated, the permissions in the entry are further constrained by the **unauthenticated** mask (described later in this section). All entries for authenticated principals, except **user_obj** and **other_obj** entries, are further constrained by the **mask_obj** mask (also described later in this section).

The following list describes the entry types for principals and groups, their meaning, and their entry format. All ACLs have a default cell defined in them, as referred to in the table. It is changeable, and serves to define the cell for various data types.

The list uses following syntax variables:

- *principal_name* is the name of a principal in the registry database.
- *group_name* is the name of a group defined in the registry database.
- *cell* is the global path name of a cell in the format */.../name*.
- *permissions* are the permissions made available by the object's ACL Manager.

Entry Type	Description
user_obj	Establishes permissions for the object's real or effective user. An example is the owner of a file. The entry format is: **{user_obj** *permissions***}**.
group_obj	Establishes permissions for members of the object's real or effective group. An example is the group of a file. The entry format is: **{group_obj** *permissions***}**
other_obj	Establishes permissions for all other principals in the default cell, unless they are specifically named in ACLs of entry type **user**, are members of a group named in an ACL with an entry type of **group**, or match the principal indicated by the **user_obj** or **group_obj** entry. The entry format is: **{other_obj** *permissions***}**
user	Establishes permissions for a specific principal in the default cell of the ACL. This ACL entry type requires a key that is a principal name. The entry format is: **{user** *principal_name permissions***}**.
group	Establishes permissions for members of a specific group in the default cell. This ACL entry type requires a key that is a group name. The entry format is: **{group** *group_name permissions***}**
foreign_user	Establishes permissions for a specific principal in a foreign cell, one other than the default cell of the ACL. You must identify the principal by supplying a principal name and cell name as a key. The entry format is: **{foreign_user** *cell_name/principal_name permissions***}**.
foreign_group	Establishes permissions for a specific group in a foreign cell, one other than the default cell of the ACL. You must identify the group by supplying a group name and a cell name as a key. The entry format is: **{foreign_group** *cell_name/group_name permissions***}**.
foreign_other	Establishes permissions for other principals in a specific foreign cell, one other than the default cell of the ACL, that are not specifically named in ACL entries of entry type **foreign_user** or are members of a group named in an ACL entry of type **foreign_group**. You must identify the foreign cell by supplying a cell name as a key. The entry format is: **{foreign_other** *cell_name permissions***}**.
any_other	Establishes permissions for all other principals in local or foreign cells unless they match a more specific entry in the ACL. The entry format is: **{any_other** *permissions***}**.

user_obj_delegate
Establishes permissions for an intermediary acting for the object's real or effective user.

The entry format is:
{user_obj_delegate *permissions***}**.

group_obj_delegate
Establishes permissions for an intermediary acting for members of the object's real or effective group.

The entry format is:
{group_obj_delegate *permissions***}**.

other_obj_delegate
Establishes permissions for an intermediary acting for all other principals in the default cell, unless they are specifically named in ACLs of entry type **user**, are members of a group named in an ACL with an entry type of **group**, or match the principal indicated by the **user_obj** or **group_obj** entry.

The entry format is:
{other_obj_delegate *permissions***}**.

user_delegate Establishes permissions for an intermediary acting for a specific principal in the default cell of the ACL. This ACL entry type requires a key that is a principal name.

The entry format is:
{user_delegate *principal_name permissions***}**.

group_delegate Establishes permissions for an intermediary acting for members of a specific group in the default cell. This ACL entry type requires a key that is a group name.

The entry format is:
{group_delegate *group_name permissions***}**.

foreign_user_delegate
Establishes permissions for an intermediary acting for a specific principal in a foreign cell, one other than the default cell of the ACL. You must identify the principal by supplying a principal name and cell name as a key.

The entry format is:
{foreign_user_delegate *cell_name/principal_name permissions***}**.

foreign_group_delegate
Establishes permissions for an intermediary acting for a specific group in a foreign cell, one other than the default cell of the ACL. You must identify the group by supplying a group name and a cell name as a key.

The entry format is:
{foreign_group_delegate *cell_name/group_name permissions***}**.

foreign_other_delegate
Establishes permissions for an intermediary acting for other principals in a specific foreign cell, one other than the default cell of the ACL, that are not specifically named in ACL entries of entry type **foreign_users** or are members of a group named in an ACL entry of type **foreign_group**. You must identify the foreign cell by supplying a cell name as a key.

The entry format is:
{foreign_other_delegate *cell_name permissions***}**.

any_other_delegate
Establishes permissions for an intermediary acting for all other principals in local or foreign cells unless they match a more specific entry in the ACL.

The entry format is:
{any_other_delegate *permissions***}**.

Group Permissions and Project Lists

Principals accrue group permissions from their project list, a list of all the groups of which a principal or alias is a member. When a principal tries to access an object, the principal has the access rights that accrue from the logical OR of permissions granted to every group with an entry in the ACL and in which the principal is a member. Note that the principal accrues rights only from the name or alias with which the principal logged in, not both names and aliases. See "Creating and Maintaining Principals, Groups, and Organizations" on page 1-48 for more information on aliases and project lists.)

For example, suppose an ACL contains the following entries:

```
{user_obj crwxid-}
{group_obj crwx---}
{other_obj -r-----}
{group composers crwx---}
{user bach crwx---}
{user mozart crwx---}
{group performers --w-idt}
```

User **cole** is a member of the group **composers** and the group **performers**. Because **cole** accrues permissions from both groups, his access permissions are **crwxidt**. (The DCE Security Service provides a method to prevent a group from being included in a project list, thus preventing the group's permissions from being accrued as part of the project list. See "Creating and Maintaining Principals, Groups, and Organizations" on page 1-48 for more information.)

Using Principal and Group ACL Entries

When a security mechanism applies ACLs, the ACL entries are chosen in a particular order. The most specific ones are chosen before the less specific.

In using the ACL entry types for principals and groups, think of the **user_obj**, **group_obj**, and **other_obj** types as being similar to the POSIX file permissions of **user**, **group** and **other**. Use the **user** and **group** types to specify permissions for a specific principal or group.

The **user_obj**, **group_obj**, **other_obj**, **user**, and **group** entry types apply to principals and groups in the default cell of the ACL. To set permissions for specific principals and groups in a foreign cell, use the **foreign_user** and **foreign_group** entries. These entries set permissions in a foreign cell in the same way that **user** and **group** entries do in the default cell. Use **foreign_other** to set permissions for others in the foreign cell, in the same way that **other_obj** does for others in the default cell.

The **any_other** entry type sets permissions for all local and foreign principals to which the other entry types do not apply. If any of the other types of entries are set for a local or foreign principal either explicitly or implicitly, the **any_other** entry will not be applied. This occurs because, after the Manager finds a match between a principal and an entry, it stops examining the ACL list and applies the found entry (or in the case of groups, entries). All other ACL entry types, except for mask types (described below), are examined by the ACL Manager to see if a match exists before the ACL Manager examines the **any_other** entry type. See "Checking Sequence for ACL Entries" in this section for details on the order of ACL checking.

ACL Entry Types for Masks

Masks in ACL entries establish maximum permissions that can be granted to a principal. There are two masks — the **mask_obj mask** and the **unauthenticated mask**. Only permissions given in an ACL entry and the mask are granted. For example, if the ACL entry specifies **rwx** permissions and the mask

specifies only the **x** permission, the permissions are ANDed with the mask, and only the **x** permission is granted.

The **mask_obj** mask, if it exists, applies to all entry types except **user_obj** and **other_obj**. The **unauthenticated** mask is applied to all unauthenticated principals. As the ACL Manager derives the permissions from the ACL entries, it filters each one through the **mask_obj** mask (if one exists), and finally through the **unauthenticated** mask. The Manager grants only those permissions that are in the first matching entry, the **mask_obj** mask, and the **unauthenticated** mask.

Note: If you do not create an **unauthenticated** mask, unauthenticated principals are denied all access to objects. If a user is unauthenticated because that user has no DCE credentials, then the only entry that the user matches is the **any_other** entry type, which is then masked by the **unauthenticated** mask. This means that for such unauthenticated users to have any access to an object, the object's ACL must contain an **any_other** type entry and an **unauthenticated** mask entry.

For a particular object, there are a great number of ACL entries specifying **rw** access to that object. You need to restrict the access to read-only, temporarily, but do not want to change all the ACL entries. Simply creating a **mask_obj** mask of **r**, and then removing it when you are done, provides the temporary restriction.

ACL Entry Types for Dissimilar DCE Releases

The **extended** entry type provides a generic format for ACL entries that allows future DCE releases to implement new ACL entry types. Because the new types are packaged in the generic format of the **extended** entry, earlier DCE releases can copy, display, and print the new entry types even if they cannot interpret their meaning.

"Copying ACLs" on page 7-45 provides instructions for copying extended entries. Note that extended entries cannot be modified; however, they can be deleted.

An **extended** ACL entry has the following form:

{extended *uuid.ndr.ndr.ndr.ndr.number_of_bytes.data permissions***}**

where:

extended	Is the ACL entry type.
uuid	Specifies a UUID that identifies the entry type of the extended ACL entry. (This UUID can identify one of the ACL entry types described in this document or an as-yet-undefined ACL entry type.)
ndr.ndr.ndr.ndr	Is a Network Data Representation (NDR) format label (in hexadecimal format and separated by dots) that identifies the encoding of data.
number_of_bytes	Is a decimal number that specifies the total number of bytes in data. It is followed by a dot.
data	Is the ACL data in hexadecimal format. (Each byte of ACL data is two hexadecimal digits.) The ACL data includes all of the ACL entry specification except the permissions. The ACL data is not interpreted; it is assumed that the ACL Manager to which the data is being passed can understand that data.
permissions	Are the permissions to be granted by the entry.

Checking Sequence for ACL Entries

An ACL Manager reads through a list of ACL entries to find the particular entry that applies to an individual who is trying to perform a particular operation. The ACL Manager first looks for a match

between the privilege attributes of the principal or process desiring access and the privilege attributes listed in the ACL. When the ACL Manager finds a match, it examines the permissions in the matching ACL entry and applies the **mask_obj** mask to it (unless it is an entry of type **user_obj** or **other_obj**) if a **mask_obj** mask exists. Finally, the ACL Manager applies the **unauthenticated** mask (if it exists) if the principal is not authenticated. If the permissions that result grant the requested access, the Manager grants it to the principal. If not, access is denied.

Because an ACL Manager stops checking the ACL entries when it finds a match, it is important to understand the order in which the ACLs are checked. Figure 7-1 on page 7-7 shows the order of checking and the masks applied. ACL Managers check entries in the following order, with the exception that the initiator principal is not checked against **..._delegate** entries. Delegate principals are checked against all entries.

1. First, the ACL Manager checks the user ACL entries, in the following order:

   a. **user_obj**
   b. **user_obj_delegate**
   c. **user**
   d. **user_delegate**
   e. **foreign_user**
   f. **foreign_user_delegate**

   The ACL Manager stops all entry checking at the first matching user entry it finds and applies the permissions in the entry. The user entries are checked in order as shown in the previous list from most specific to least specific.

2. If the ACL Manager does not find a match in the user entries, it checks *all* of the following group entries:

   a. **group_obj**
   b. **group_obj_delegate**
   c. **group**
   d. **group_delegate**
   e. **foreign_group**
   f. **foreign_group_delegate**

   If any group ACL entries match the principal's project list, and the logical OR of permissions from these entries grants access, then access is granted and no further checking is performed.

   Because principals accrue permissions from all groups listed in the ACL of which they are a member (and for which they are in the project list), *all* the groups are checked and *all* the principal's group permissions are logically ORed. The order of group entry checking is not important. See “Group Permissions and Project Lists” in this section for more information on project lists.

3. If the ACL Manager does not find a match between the principal requesting permission and a member of a group in the group entries, it checks the **other_obj** and **other_obj_delegate** entries. If the ACL Manager finds a match, it stops checking ACL entries.

4. If the ACL Manager does not find a match between the principal requesting permission and the **other_obj** or **other_obj_delegate** entries, it checks the **foreign_other** and **foreign_other_delegate** entries. If the ACL Manager finds a match, it stops checking ACL entries.

5. If the ACL Manager does not find a match between the principal requesting permission and the **foreign_other** or **foreign_other_delegate** entries, it checks the **any_other** and **any_other_delegate** entries. If it does not find a match in the **any_other** or **any_other_delegate** entries, it denies all access to the object.

The final permission is the intersection of the permission of the initiator principal and of each delegate.

Figure 7-1 on page 7-7 shows these steps as they apply to the ACL entries. The two columns distinguish between ACL entries that are not masked by **mask_obj** and those that are masked by it.

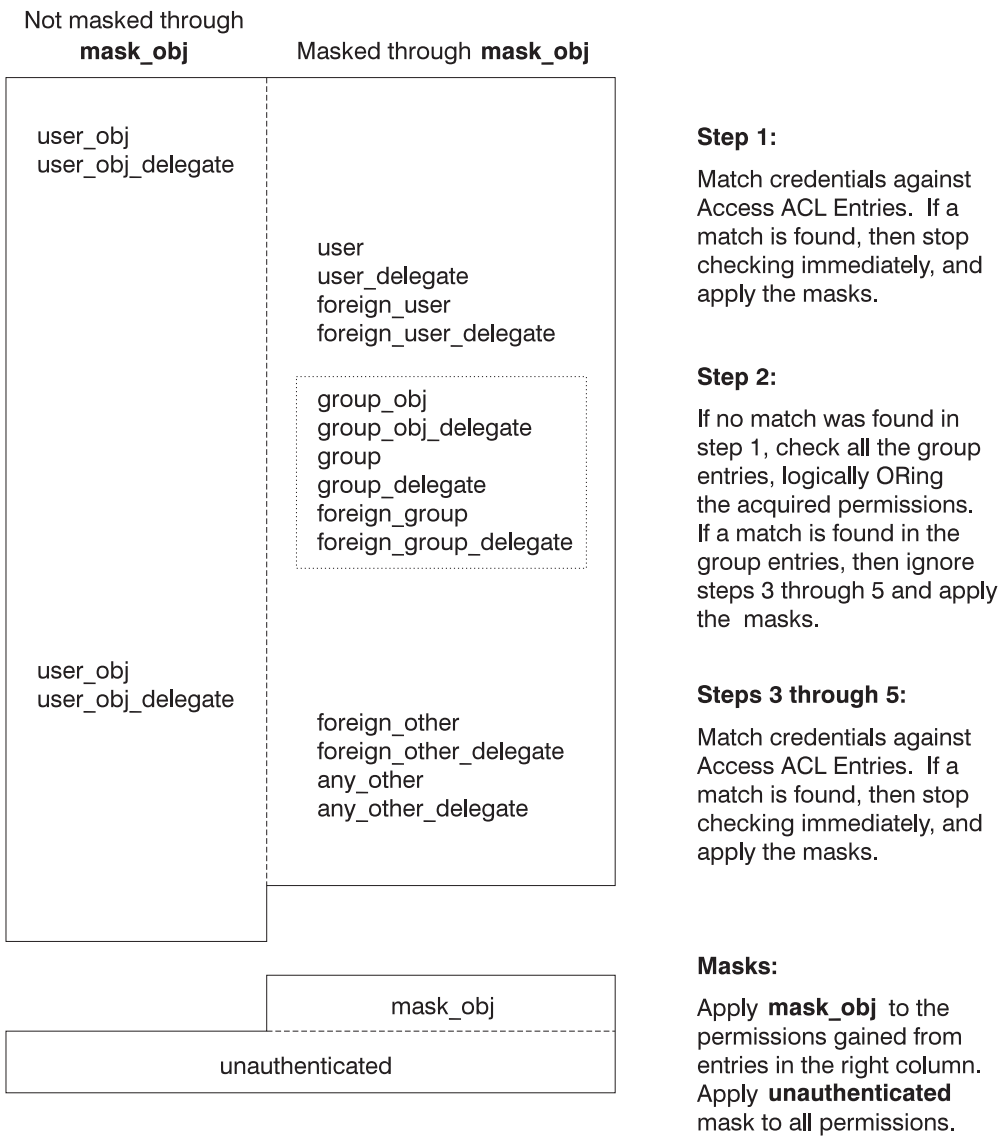


Figure 7-1. Order of Checking ACLs and Applying Masks

The mask_obj *Mask and ACL Checking*

Before the ACL Manager grants any permissions derived from checking the ACL entries, it filters the entry permissions through the **mask_obj** mask. Only those permissions named in the ACL entry and in the mask are granted. For example, if an ACL entry grants **rwx** permissions and the **mask_obj** entry specifies only **r** and **w** permission, only **r** and **w** are granted. The **x** permission named in the ACL entry is ignored.

The Unauthenticated Mask and ACL Checking

If an ACL Manager receives an access request from an unauthenticated principal, it checks the ACL entries and applies the **mask_obj** mask, if available, as described previously. It then filters the resulting permissions through the mask for unauthenticated principals (entry type of **unauthenticated**). Only those permissions specified in the **unauthenticated** mask, in the ACL entry, and in the **mask_obj** mask, if it exists, are granted.

The Effect of the Checking Order on Granting Permissions

You can think of the order in which the ACL entries are checked as going from most specific to least specific. For example, assume an ACL contains the following entries:

```
{user mahler r}
{group composers rwx}
```

If the principal named **mahler**, who is a member of the group **composers**, requests execute (**x**) access, it is denied. This happens because the order of checking specifies that all user entries (**user_obj**, **user**, and **foreign_user**) are checked before all group (**group_obj**, **group**, and **foreign_group**) entries. Therefore, the first match found by the ACL Manager is the match between user **mahler** and the ACL entry for user **mahler**. When a matching user entry is found, checking stops and the found permissions are applied. In this case, checking stops before the **group** entry, the entry with the more liberal permissions.

Denying Access

When you create an ACL entry for a principal or group, you grant only the permissions you specify in the ACL entry. To deny a principal all access to an object, create an ACL entry that contains a dash in place of the permissions. For example to deny all access to user **mozart**, the entry would be:

```
{user mozart -}
```

If you choose to deny access to a specific principal or group, select the most specific entry type available. Generally for principals this is an entry type of **user** or **foreign_user**; for groups, it is an entry type of **group** or **foreign_group**. Note that if the principal is the object's owner or a member of the object's group, you must use the **user_obj** or **group_obj** entry types to ensure that access is denied.

To deny access to all unauthenticated users, do not create the **unauthenticated** mask. If this mask is not created (ACL entry type of **unauthenticated**), only authenticated principals can access the object. The same behavior is achieved by creating an **unauthenticated** mask with no permissions (or a dash in place of the permissions). This method also has the additional advantage of illustrating graphically that unauthenticated users have no access rights.

Examples — ACL Entries and Masks

The three sample ACL entries in Figure 7-2 are in the format that Tcl accepts for input.

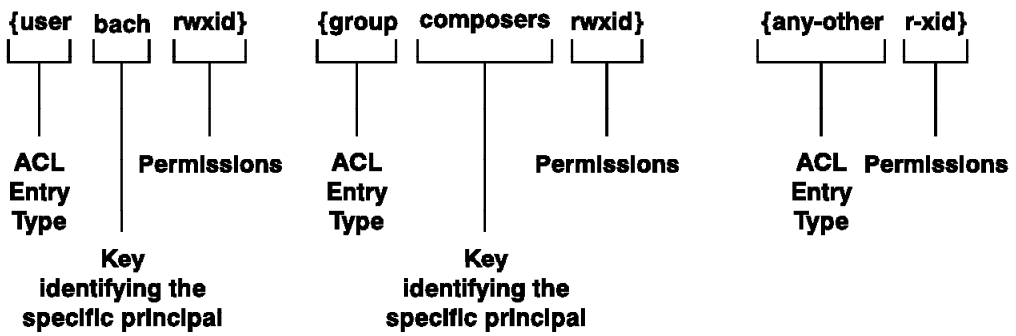


Figure 7-2. Sample ACL Entries

The first sample ACL entry sets permissions for a principal in the local cell, named **bach**. The ACL entry type is **user**, the key is **bach,** and the permissions are **rwxid**. The entry components are separated by the space character.

The second sample ACL entry sets permissions for a group in the local cell, named **composers**. The ACL entry type is **group**, the key is **composers,** and the permissions are **rwxid**.

The third sample ACL entry sets permissions for all other principals in the local cell or foreign cells (unless they match a more specific entry). The ACL entry type is **any-other**, there is no key, and the permissions are **r-xid**. Not all types of ACL entries require a key.

On output, the Tcl format for ACL permissions contains either a permission character or a – (dash) for each possible permission. Two examples are:

```
{user mozart crwx---}
{user brahms -------}
```

For input, the output format is acceptable, or you can use a relaxed form that omits the dashes. For input, the same examples can be shortened to:

```
{user mozart crwx}
{user brahms -}
```

The single dash is retained to show that user **brahms** is denied all permissions.

ACL Management Tasks

ACL management involves creating, modifying, and deleting the entries for the ACLs on DCE entities. You can use the DCE control program to do all of these tasks. The control program's **acl** commands perform the following operations on ACLs:

- Create and modify ACL entries for DCE objects in the local cell and foreign cells. (Note that when objects are created they are associated with initial ACL entries. See “Objects and Containers” under “More Detailed Information — Container ACLs” on page 7-39 for more information.)
- Display the permissions implemented for an object by the object's ACL Manager.
- Create and modify masks used to restrict allowable permissions.

Note: Standard AIX tools that display and manipulate AIX modes have an effect only on the ACLs established for the file system.

For a detailed description of the DCE control program's **acl** commands, see the *DCE for AIX Administration Command Reference*.

Adding Accounts

Introduction — Adding Accounts

Use the **dcecp account create** command to add accounts to the registry. Information that is associated with accounts falls roughly into the following two categories:

- User information similar to that typically found in the /**etc**/**passwd** file.
- Authentication policy that lets you control the account's access to the network. Authentication policy establishes account and password validity, account expiration policy, and ticket expiration policy. The tighter you control authentication policy, the more secure your cell is, but the more processing overhead you can accrue.

Both types of information are supplied as attributes in standard **dcecp** attribute lists or as attribute options.

More Detailed Information — Adding Accounts

Note that authentication policy can also be set for the registry. If the registry policy differs from the policy that you enter for an account, the stricter policy applies. See “Handling Conflicting Policies” under “More Detailed Information — Authentication Policy” on page 7-33 for more information on contradictory policy.

The following list describes the attribute options used to create accounts. Note that the options described in this table can also be supplied without the hyphens in attribute lists.

Option	Meaning
–acctvalid {yes|no}	A flag that determines account validity. If you set this flag to **no**, the account is invalid and the account principal cannot log into the account. The default is **yes**.
–client {yes|no}	A flag that indicates whether the account is for a principal that can act as a client. If you set this flag to **yes**, the principal is able to log into the account and acquire tickets for authentication. The default is **yes**.
–description *string*	A text string in PCS format that is typically used to describe the use of the account. No default.
–dupkey {yes|no}	A flag that determines if tickets issued to the account's principal can have duplicate keys. The default is **no**.
–expdate	The date (in the ISO timestamp format YY-MM-DD-hh:mm:ss) the account expires. To renew an account after it expires, change the date. The default is **none**, meaning the account never expires.
–forwardabletkt {yes|no}	A flag that determines whether a new ticket-granting ticket with a network address that differs from the present ticket-granting ticket's network address can be issued to the account's principal. (The **–proxiabletkt** attribute performs the same function for service tickets.) The default is **yes**.
–goodsince *date*	The date and time (in the ISO timestamp format YY-MM-DD-hh:mm:ss) that the account was last known to be in an uncompromised state. Any tickets granted before this date are invalid. Control over this date is especially useful if you know that an account's password was compromised. Changing the password can prevent the unauthorized principal from accessing the system again by using that password, but it does not prevent the principal from accessing the system components that tickets were obtained for fraudulently before the password was changed. To eliminate the principal's access to the system, the tickets must be cancelled. If you set the **–goodsince** attribute to the date and time the compromised password was changed, all tickets issued before that time are invalidated and the unauthorized principal's system access is eliminated. When the account is initially created, the **–goodsince** attribute is set to the current date.
–group *group_name*	The name of the group that is associated with the account. This attribute must be supplied to create an account; there is no default.
–home *dir_name*	The directory in which the principal is placed at login. There is no default.
–organization *org_name*	The name of the organization that is associated with the account. This attribute must be supplied to create an account; there is no default.
–password *password*	The required password for the account in plain text. The system encrypts the password you supply. There is no default.

–postdatedtkt {yes|no}
A flag that determines whether tickets with a start time in the future can be issued to the account's principal. The default is **no**.

–proxiabletkt {yes|no}
A flag determines whether a new ticket with a different network address than the present ticket can be issued to the account's principal. (The **–forwardabletkt** attribute option performs the same function for ticket-granting tickets.) The default is **no**.

–pwdvalid {yes|no} A flag that determines whether the current password is valid. If this flag is set to **no**, the account password has expired and the principal is prompted to change it the next time the principal logs into the account. The default is **yes**.

–renewabletkt {yes|no}
The Kerberos V5 renewable ticket feature is not currently used by the DCE; any use of the renewable ticket attribute is unsupported at the present time.

–server {yes|no} A flag that indicates whether the account is for a principal that can act as a server. If the account is for a server that engages in authenticated communications, set this flag to **yes**. The default is **yes**.

–shell *path_to_shell* The shell that runs when a principal logs in.

–stdtgtauth {yes|no} A flag that determines whether tickets issued to the account's principal can use the ticket-granting ticket authentication mechanism. The default is **yes**.

–maxtktlife *hours* The maximum ticket lifetime. This is the maximum amount of time (in hours) that a ticket can be valid. When a client requests a ticket to a server, the lifetime granted to the ticket takes into account the **maxtktlife** attribute value for both the server and the client. In other words, the lifetime cannot exceed the shorter of the server's or client's maximum ticket lifetime.

If you do not specify a **maxtktlifetime** attribute value for an account, the **maxtktlifetime** attribute value defined for the registry authorization policy is used.

–maxtktrenew *hours* The maximum ticket renewable. This is the amount of time in hours before a principal's ticket-granting ticket expires and that principal must log into the system again to reauthenticate and obtain another ticket-granting ticket. The lifetime of the principal's service tickets can never exceed the lifetime of the principal's ticket-granting ticket. The shorter you make Maximum Certificate Renewable, the greater the security of the system. However, since principals must log in again to renew their ticket-granting ticket, the time needs to take into consideration user convenience and the level of security required.

If you do not specify a **maxtktrenew** attribute value for an account, the **maxtktrenew** attribute value defined for the registry authorization policy is used.

Note: The maximum ticket lifetime and maximum ticket renewable can be set as registry properties for the registry as a whole with the **dcecp registry modify** command. When they are set with the **dcecp account create** or **account modify** commands, they apply only to a specific account.

Setting Ticket Lifetimes

You should be aware of two other options set by the **dcecp registry modify** command; default ticket lifetimes and minimum ticket lifetime:

- **Minimum Ticket Lifetime** — The shortest possible lifetime that can be assigned to a ticket. Note that the actual effective value of Minimum Ticket Lifetime is affected by Default Certificate Lifetime.

- **Default Ticket Lifetime** — The lifetime granted for tickets, unless the principal specifically requests a different lifetime. Although principals can request a specific lifetime for a ticket, the majority accept the default lifetime. (If a principal requests a ticket lifetime of 0 (zero), the default lifetime is assigned to the ticket.)

  Note that the actual effective value of Default Ticket Lifetime is affected by Maximum Certificate Lifetime.

The actual lifetimes assigned to tickets depends on rules enforced by the Security service regarding the settings of Maximum Ticket Lifetime, Default Ticket Lifetime, and Minimum Ticket Lifetime. These rules are as follows:

- The maximum ticket lifetime can never be larger than the renewable ticket lifetime (in other words, max_life = min (max_life, renewable_life)) or less than 60 seconds. If the maximum ticket lifetime is larger than the renewable ticket lifetime, then the renewable ticket lifetime is used as the maximum ticket lifetime. For example, suppose an account's is set to 15 hours. If you set the renewable ticket lifetime to 20 hours, the effective maximum ticket lifetime is not 20, but 15 hours.
- The default ticket lifetime can never be larger than the maximum ticket lifetime (in other words, default_life = min (default_life, max_life)) or less than 60 seconds. If the default ticket lifetime is larger than the maximum ticket lifetime, then the maximum ticket lifetime is used as the default ticket lifetime. For example, suppose registry policy specifies a default ticket lifetime of 25 hours. If you set the registry's maximum ticket lifetime to 15 hours, the registry's effective default certificate lifetime is not 25, but 15 hours.
- The minimum ticket lifetime can never be larger than the default certificate lifetime (in other words, min_life = min (min_life, default_life)) or less than 60 seconds. If the minimum ticket lifetime is larger than the default certificate lifetime, then the default ticket lifetime is used as the minimum ticket lifetime. For example, suppose registry policy specifies a default ticket lifetime of 10 hours. If you set an account's minimum ticket lifetime to 15 hours, the account's effective minimum ticket lifetime is not 15, but 10 hours.

Although **dcecp** lets you enter values contrary to the rules and displays these values when you view the account's policies (with the **account show** command), the values used are the ones described in the rules, not the ones you entered.

Note: To be exact, clocks in the network must be synchronized for the times that are associated with registry data.

Ticket-Granting Ticket Lifetimes and Service Ticket Lifetimes

The Authentication Service never grants a principal a service ticket with a lifetime that exceeds the time remaining in the principal's ticket-granting ticket lifetime. For example, if 2 hours remain in the life of a principal's ticket-granting ticket and the principal requests or accepts a default of 4 hours for a service ticket's lifetime, only the 2-hour lifetime is granted.

If the renewable ticket flag (the **renewabletkt** attribute) is set on for a principal's account, the lifetime of the principal's ticket-granting ticket also affects the renewal of service tickets. No service ticket is renewed with a lifetime that exceeds the remaining lifetime of the principal's ticket-granting ticket. Service tickets are normally renewed for the lifetime that is allocated to the original ticket. If the original time exceeds the lifetime of the ticket-granting ticket, the ticket is renewed only for the time remaining to the ticket-granting ticket.

Modifying Accounts

The **dcecp account modify** command with the allows you to modify accounts. You can modify any of the account attributes.

When you modify accounts, you must supply your password with the **–mypwd** option to verify your identity. Note that the password you type is not displayed on the screen. If you do not enter a password, you are prompted for it. Because you are required to enter your password, you must run the command in interactive mode. You cannot run it in command-line mode where your password cannot be prevented from displaying on the screen.

The following example shows how to use the **account modify** command to specify a new home directory for **mahler**'s account:

```
dcecp> account modify mahler -home /.../music/fs/users/mahler/concert -mypwd password
dcecp>
```

Note that you can also use the **–change** option with **account modify** to supply the changes in an attribute list. The **–add** and **–remove** options are not supported with **account modify** command because each account attribute must be present and must have a value.

Deleting Accounts

The following example illustrates the use of the **dcecp account delete** command to delete the account for the principal **mahler**.

```
dcecp> account delete mahler
```

If you delete a group or organization, you will also automatically delete any accounts that are associated with that group or organization.

You can delete multiple accounts with one **account delete** command. To do so enclose the names of the account principals in braces, separated by spaces. For example, to delete accounts for **bach**, **britten**, and **mahler**, you would enter:

```
dcecp> account delete {bach britten mahler}
dcecp>
```

Use the **dcecp account create** command to create accounts. When you use the **account create** command, you must supply the name of the principal for which the account is being created and the group and organization with which the account is associated. In addition, you must supply your password with the **–mypwd** option to verify your identity. If you do not enter your password, **dcecp** will prompt you. All other attributes can be allowed to default. Note that the password you type is not displayed on the screen.

Because you are required to enter your password, you must run the command in interactive mode. You cannot run it in command-line mode where your password cannot be prevented from displaying on the screen.

Steps — Adding Accounts

To add new user accounts to the registry, you must have the appropriate permissions to the registry database (see “Accessing Registry Objects” on page 1-44). After establishing the appropriate permissions, you can proceed to add accounts as follows:

1. If the principal to be used in the account does not already exist, enter the **principal create** command to add the principal.
2. Enter the **group create** command to add the group to be used in the account if this group does not already exist.
3. Enter the **organization create** command to add the organization to be used in the account if this organization does not already exist.

4. Finally, enter the **account create** command to add the account.

Examples — Adding Accounts

- The following example shows the **dcecp account create** command used to create an account for the principal **mahler**, which is associated with the group **symphonists** and the organization **classic**. All other account attributes are allowed to default.

```
dcecp> account create mahler -group symphonists -organization classic -mypwd password
dcecp>
```

- It is possible to create multiple accounts with one **account create** command. To do so enclose the names of the principals for whom the accounts are being created in braces, separated by spaces. For example, to create accounts for the principals **bach**, **britten**, and **mahler**, you could enter:

```
dcecp> account create {bach britten mahler} -group symphonists -organization classic
                    -password music -mypwd password
dcecp>
```

  Note: When you create multiple accounts each account is assigned the same attributes. This means that in the example above the accounts for **bach**, **britten**, and **mahler** are all associated with the **symphonists** group and **classic** organization, and they are all assigned the password **music**. You may find it useful to create multiple accounts this way for principals that all belong to the same group and organization. Notify users whose accounts were created this way to change their passwords immediately.

Adding and Maintaining Groups and Organizations

Introduction — Adding and Maintaining Groups and Organizations

A group or organization must have been added to the registry before it can be used in an account. When you use the **dcecp group create** command to add groups, you can set a project list inclusion property that controls whether individual groups are included in project lists. (Project lists do not apply to organizations.)

More Detailed Information — Adding and Maintaining Groups and Organizations

Project Lists

A principal's project list identifies all the groups of which a principal or alias is a member. When a principal tries to access an object, the principal has the access rights that accrue from membership in every group that is named in the object's Access Control List (ACL). See the “Using Access Control Lists” on page 1-59 for a description of ACLs. For example, the ACL for file X contains two entries — one permits group A write access and one permits group B read access. Any principal who is a member of both groups A and B can read and write to file X.

Principals accrue project list access rights only from the groups that are associated with the name or alias with which they log in. They do not accrue rights from their names and all of their aliases. For example, consider the case where a principal named **gustav** is a member of groups A and B and this same principal has an alias (**gus**) that is a member of groups C and D. When the principal logs in as **gustav**, the principal accrues access rights from groups A and B only. When the principal logs in with the alias **gus**, the principal accrues access rights from groups C and D only.

To display the groups of which a principal (or its alias) is a member, use the **principal show** command as described in “Displaying Registry Information” on page 1-57.

Prohibiting Inclusion on Project Lists

If a group is prohibited from inclusion in a project list, its rights are not accrued. For example, assume again that file X's ACL includes two entries — one that permits group A read access to file X and one that permits group B write access to file X. Assume that the project list inclusion property is set to disallow group B from project lists. A principal who is a member of both groups A and B who tries to access file X is allowed only read permission, not write permission. If the project list inclusion property allows group B to be on project lists, a member of groups A and B receives both read and write access.

You may decide to prohibit some groups from inclusion on the list. You may, for example, want to prohibit any reserved groups with access rights similar to **root** from inclusion on any project lists.

Adding Groups and Organizations

Use the **dcecp group create** command to add groups and the **dcecp organization create** command to add organizations. When you add a group or organization, you must specify the group's or organization's primary name. In addition, you can supply the attribute options listed below.

When you use the **dcecp group create** command and the **dcecp organization create** command, you can create multiple groups or organizations with one command in the same way that you can create multiple principals. See “Adding Principals” under “More Detailed Information — Adding and Maintaining Principals” on page 7-17 for details.

Information	Meaning
–gid	The required AIX ID that is associated with the group or organization. You can enter this number explicitly or allow it to be generated automatically. The number that is entered cannot exceed the maximum allowable AIX number (the **maxuid** attribute) set with the **dcecp registry modify** command; however, you can enter a number lower than the minimum AIX number (the **minuid** attribute) set for groups or organizations with the **registry modify** command. If you allow the number to be assigned automatically, it falls in the range defined by the minimum AIX number and the maximum AIX number.
–fullname *string*	An optional name that is used to more fully describe a primary name. To include spaces, enclose the full name in braces. The default is blank.
–inprojlist *value*	For groups only, whether the group can be on project lists, **yes** is the default.

Modifying Groups and Organizations

For groups and organizations, you can change the primary name and full name. In addition, for groups you can change whether or not the group can appear in project lists; for organizations you can change policy. See “Maintaining Policies and Properties” on page 1-58 for details changing organization policy.

Use the **dcecp group modify** command to modify change groups.

Use the **dcecp group rename** command to change a group's primary name or the **dcecp organization rename** command to change an organization's primary name. These commands have the form:

group rename *old_name* **-to** *new_name* **organization rename** *old_name* **-to** *new_name*

where:

old_name Is the primary name of the group or organization to be changed.

new_name Is the new primary name of the group or organization.

If you change a primary name, that change is reflected in the membership lists of all groups and organizations in which the group or organization is listed as a member.

Deleting Groups and Organizations

If you delete a group or organization, you also automatically delete any accounts that use the group or organization. For example, if you delete the group **symphonists**, you also automatically delete the accounts **vivaldi symphonists baroque** and **mozart symphonists classic**.

Use the **dcecp group delete** to delete groups and the **dcecp organization delete** command to delete organizations.

You can delete multiple groups or organizations with a single **group delete** or **organization delete** command by including the names to delete in braces and separated by spaces just as you would to delete multiple principals.

Examples — Adding and Maintaining Groups and Organizations

- The following example shows how to add a group named **symphonists** to the registry.

  ```
  dcecp> group create symphonists
  dcecp>
  ```

  In the example, the group AIX ID is generated automatically, the full name is not supplied, and the group is included on project lists.

- The following example shows how to add an organization named **classic** to the registry.

  ```
  dcecp> organization create classic
  dcecp>
  ```

  In the example, the organization AIX ID is generated automatically and the full name is not supplied.

- The following example shows the use of this command with the **–inprojlist** option to change the group symphonist's project list inclusion property from **yes** (include on project lists) to **no** (prohibit from project lists).

  ```
  dcecp> group modify symphonists -inprojlist no
  dcecp>
  ```

- The following example shows the **group rename** command used to change a full name from **symphonists** and **symphonists7**.

  ```
  dcecp> group rename symphonists -to symphonists7
  dcecp>
  ```

- The following example shows the **group delete** command being used to delete the group **symphonists**.

  ```
  dcecp> group delete symphonists
  dcecp>
  ```

- The following example shows the **organization delete** command being used to delete the organization **classic**.

  ```
  dcecp> organization delete classic
  dcecp>
  ```

Adding and Maintaining Principals

Introduction — Adding and Maintaining Principals

Use the **dcecp principal create** command to create principals. A principal must exist before you can create an account for the principal.

More Detailed Information — Adding and Maintaining Principals

When you use the **dcecp principal create** command, you must supply the principal's primary name as an argument. In addition, you can supply the attribute options summarized in the following list.

Option	Meaning
–fullname *namestring*	An optional name that is used to more fully describe a primary name. To include spaces, enclose the full name in braces. The default is blank.
–uid *integer*	The required AIX ID that is associated with the principal. You can enter this number explicitly or allow it to be generated automatically. If you enter it, the number you enter cannot exceed the maximum allowable AIX number (the **maxuid** attribute) set with the **registry modify** command. You can, however, enter a number lower than the minimum AIX number (the **minuid** attribute) set for principals with the **registry modify** command. If you allow the number to be assigned automatically, it falls in the range defined by the minimum AIX number and maximum AIX number.
–quota *quota*	The number of registry objects that can be created by the principal, known as the principal's object-creation quota. To allow a principal to create an unlimited number of registry objects, enter the text string `unlimited` to set no quota. To prevent a principal from creating any registry objects, enter a 0.*quota*, which defaults to `unlimited`.

Note: In addition to these standard principal attributes, you can also attach Extended Registry Attribute (ERA) instances to principals to control such aspects of DCE security as preauthentication, password strength and password generation, and handling of invalid logins. See “Extended Security Attributes for Principals” on page 7-79 for information on these well-known ERAs. See “Creating and Using Extended Registry Attributes” on page 1-49 for information on ERAs in general.

Adding Principals

To add principals to the registry, use the **principal create** command.

It is possible to create multiple principals with one **preferred create** command. To do so, enclose the principal names in braces, separated by spaces.

If you create multiple principals, you must allow the principal's AIX ID to default to the system-assigned ID. This is because, if you include an attribute option in the command line, that attribute value is assigned to each principal.

Changing Principals

You can change a principal's primary name and other information that is related to the principal. Additionally, you can change a primary name to an alias and an alias to a primary name. If you change a primary name to an alias and do not make an alias the primary name, operations that return names choose one of the aliases at random.

Changing Primary Names

Use the **dcecp principal rename** command to change a primary name. Enter the command in the following form:

principal rename *old_name* **-to** *new_name*

where:

old_name Is the primary name of the principal to be changed.

new_name Is the new primary name of the principal.

If you change a primary name, that change is reflected in the membership lists of all groups and organizations of which the principal is a member.

In the unusual case where you are changing a host's principal name while the host is logged into a DCE cell, the existing host credentials will become invalid unless you perform extra steps to update the host credentials with the new principal name.

Host credentials are managed by the **secval** process which performs security client functions on a DCE host. Normally, just after the host starts, the **secval** process logs the host into the DCE cell, getting the host credentials and storing them on the host. Deactivate and reactivate the **secval** process to update these credentials after changing the principal name.

Changing Principal Information

Use the **dcecp principal modify** command to change any principal information except AIX ID and User ID.

Deleting Principals and Aliases

If you delete a principal or an alias, the system automatically deletes any accounts for that principal or alias. For example, if you delete the principal **mahler**, the **mahler composers classic** account is also deleted. If you delete the principal alias **gustav**, you also delete the **gustav music_admin classic** account. Be aware that deleting a principal or a principal's alias could orphan the objects that are owned by the principal/UUID.

You can delete multiple principals or aliases with one **principal delete** command. To do so, enclose the principal names in braces, separated by spaces.

Examples — Adding and Maintaining Principals

- The following sample command creates a principal with a primary name of **mahler** and a full name of **gustav mahler**.

  ```
  dcecp> principal create mahler -fullname {gustav mahler} -quota 5
  dcecp>
  ```

  In the example, the AIX number defaults to one that is generated automatically. Notice that, because the full name (**gustav mahler**) assigned to the principal contains a space, it is enclosed in braces.

- To create the principals **bach**, **britten**, **mahler**, and **satie**, enter:

  ```
  dcecp> principal create {bach britten mahler satie}
  dcecp>
  ```

- The following sample command creates the principals **bach**, **britten**, **mahler**, and assigns each object a creation quota of **5**.

  ```
  dcecp> principal create {bach britten mahler satie} -quota 5
  ```

- The following example shows the **principal rename** command used to change a full name from **mahlar** to **mahler**.

```
dcecp> principal rename mahlar -to mahler
dcecp>
```

- The following example illustrates these operations on remote host **persephone**.

```
dcecp> secval deactivate /.:/hosts/persephone/config/secval
dcecp> secval activate /.:/hosts/persephone/config/secval
dcecp>
```

- The following example shows the **principal modify** command used to change principal **mahler**'s object creation quota to **10**.

```
dcecp> principal modify mahler -quota 10
dcecp>
```

- The following example shows the **principal delete** command used to delete the principal named **mahler**.

```
dcecp> principal delete mahler
dcecp>
```

- To delete the principals **bach**, **britten**, and **mahler**, you enter:

```
dcecp> principal delete {bach britten mahler}
dcecp>
```

Administration and Programming in DCE Audit

Introduction — Administration and Programming in DCE Audit

Many of the DCE Audit Service administrative tasks are related to the tasks performed by the application programmer. To understand these administrative tasks, you should be familiar with some programming aspects of the DCE Audit Service. This section describes a typical DCE Audit Service programming and administrative scenario and their tasks.

More Detailed Information — Administration and Programming in DCE Audit

Programmer Tasks

The application programmer uses the DCE Audit APIs to enable auditing in the application server program. Specifically, the programmer performs the following tasks:

1. Identifies the code points corresponding to the audit events in the application server program.
2. Assigns an event number to each code point. The event numbers are used as parameters by the **dce_aud_open()** API which opens an audit trail, and the **dce_aud_start()** API, which initializes the audit record for the code point. The programmer may want to define these event numbers in the server's header file.
3. Adds a call to the **dce_aud_open()** API to the application server's initialization routines. This opens the audit trail file. This function uses the event number of the lowest numbered event, in this case, **acct_open()**, as one of its parameters.
4. Adds Audit event logging functions to every code point in the application server code. These functions perform the following at each code point:
   - Initializes an audit record using the **dce_aud_start()** API. This function assigns the event number to the code point representing an event. Thus, this function uses the event number as one of its parameters.
   - Adds event-specific information to the audit record using the **dce_aud_put_ev_info()** API.

- Commits the audit record using the **dce_aud_commit()** API. This function writes the audit record to the audit trail file.

5. Closes the audit trail file when the server shuts down, using the **dce_aud_close()** API in the main server routine.

Administrator Tasks

The administrator uses the event numbers representing the different code points in the audit client application server program to create event class files and filter guides in the following manner:

1. The administrator obtains the event numbers of the code points (representing each audit event) from the application server programmer.
2. The administrator decides to create two event classes — the **account_creation_operations** class comprised of **acct_open()** and **acct_close()**, and the **account_balance_operations** class comprised of **acct_withdraw()**, **acct_deposit()**, and **acct_transfer()**. He assigns the event class account_creation_operations the event class number 0xC0000006. Event class account_balance_operations is assigned the event class number 0xC0000007.

   To create the event classes, the administrator creates and edits two files, one for each event class. The name of each of these files is the same as the event class that each represents. Each file contains the numbers of the events in each event class.
3. The administrator decides to create two filters, one for all users within the cell (for the cell /**.:/torolabcell**), and the other for all other users.

   The filter for all users within the cell has the following guidelines:

   - Audit the events in the event class **account_balance_operations** only, subject to the next condition.
   - Write an audit record only if an operation in that event class failed because of access denial.
   - If the first condition is fulfilled, write the audit record in an audit trail file only.
   - The administrator then uses the DCE control program's **audfilter create** command to create this filter.

   The filter for all other users has the following guides:

   - Audit the events in both event classes, subject to the next condition.
   - Write an audit record if an operation in that event class succeeded, failed, or failed because of access denial.
   - Write the audit record both in an audit trail file and the console.

   Following is the **dcecp** session for creating this filter:

```
dcecp> audfilter create  world -attribute
       {account_balance_operations,account_creation_operations alarm,log all}
```

   "DCE Audit Service Administrative Tasks" on page 1-49 provides detailed information about the DCE control program's **audfilter create** command.

Examples — Administration and Programming in DCE Audit

- **Programmer Tasks Example** — A banking server example illustrates a typical DCE Audit Service programming and administrative scenario.
  - Step 1 example (possible functions of a banking server program):

    A banking server program can have the following functions:

    - **acct_open()**

- **acct_close()**
- **acct_withdraw()**
- **acct_deposit()**
- **acct_transfer()**

Each of these functions can be designated as a code point, meaning these are possible audit events that can be recorded (depending on the filter):

```
......
acct_open()               /* first code point */
.....
acct_close()              /* second code point */
.....
acct_withdraw()           /* third code point */
.....
acct_deposit()            /* fourth code point */
.....
acct_transfer()           /* fifth code point */
.....
.....
```

– Step 2 example (definition of event numbers in the server's header file):

```
   /* event number for the first code point, acct_open() */
#define evt_vn_bank_server_acct_open      0x01000000
   /* event number for the second code point, acct_close() */
#define evt_vn_bank_server_acct_close     0x01000001
   /* event number for the third code point, acct_withdraw() */
#define evt_vn_bank_server_acct_withdraw  0x01000002
   /* event number for the fourth code point, acct_deposit() */
#define evt_vn_bank_server_acct_deposit   0x01000003
   /* event number for the fifth code point, acct_transfer() */
#define evt_vn_bank_server_acct_transfer  0x01000004
```

– Step 3 example (use of event number as a parameter):

```
main()
   ....
   ....
   /* evt_vn_bank_server_acct_open is the lowest event number */
   dce_aud_open(aud_c_trl_open_write, description,
                evt_vn_bank_server_acct_open,
                5, &audit_trail, &status);
   ....
```

– Step 4 example (use of APIs on the code points of the bank server program):

```
......
acct_open()     /* first code point */
.......
/* Uses the event number for acct_open(), evt_vn_bank_server_acct_open */
dce_aud_start(evt_vn_bank_server_acct_open,
              binding,options,outcome,&ard, &status);
.......
  if (ard) /* If events need to be logged */
      dce_aud_put_ev_info(ard,info,&status);
......
  if (ard) /* If events were logged */
        dce_aud_commit(at,ard,options,format,&outcome,&status);
.....
.....
.....
 acct_close()   /* second code point */
 .......
 /* Uses the event number for acct_close(), evt_vn_bank_server_acct_close */
 dce_aud_start(evt_vn_bank_server_acct_close,
```

```
                binding,options,outcome,&ard, &status);
.......
  if (ard) /* If events need to be logged */
       dce_aud_put_ev_info(ard,info,&status);
......
  if (ard) /* If events were logged */
       dce_aud_commit(at,ard,options,format,&outcome,&status);
.....
.....
.....
```

– Step 5 example (shutting down):

```
  .......
dce_aud_close(audit_trail, &status);
  .......
```

- **Administrator Tasks Example** — The administrator creates event class files and filter guides in the following manner:
  – Step 1 example (obtaining event numbers):

    These code points were assigned the following event numbers:

acctopen()	0x01000000
acctclose()	0x01000001
acctwithdraw()	0x01000002
acctdeposit()	0x01000003
accttransfer()	0x01000004

    (Note that event numbers should completely sequential. That is, no missing members of the sequence is allowed.)

  – Step 2 example (creation of two event classes):

    The file with the name **account_creation_operations** is edited as follows: (lines that begin with # are comment lines)

```
# Event class number of account_creation_operations
ECN = 0xC0000006
# Event number of acct_open()
0xC1000000
# Event number of acct_close()
0xC1000001
```

    The file named **account_balance_operations** is edited as follows:

```
# Event number of acct_withdraw()
0xC1000002
# Event number of acct_deposit()
0xC1000003
# Event number of acct_transfer()
0xC1000004
```

    The administrator stores both files in the **/opt/dcelocal/etc/audit/ec** directory.

  – Step 3 example (creation of two filters):

```
dcecp> audfilter create {cell /.../torolabcell} -attribute
                  {account_balance_operations denial log}
```

Adopting Registry Orphans

Introduction — Adopting Registry Orphans

Although the **dcecp** command displays object names and you identify registry objects by name, the Security component uses UUIDs to identify objects internally. When you create a registry object, the Security component automatically sets up an association between the object name and a UUID that it uses to identify the object. When you delete registry objects, you delete the association between the registry object and the UUID that identifies the object.

Orphans are objects owned by UUIDs that are not associated with a principal or group because the principal or group has been deleted. For example, if you delete a principal from the registry, you also delete the association between the name used to identify the principal externally and the UUID used to identify the principal internally. Any objects (files, programs) owned by the deleted principal are now owned internally by a UUID no longer associated with a principal. If no other principal, group, or organization has access rights to the object, the object cannot be accessed at all and is now an orphan.

To solve this problem, you can use the **dcecp principal create**, **group create**, and **org create** commands with the **–uuid** option to create a principal, group, or organization with the same UUID as the UUID that owns the orphaned object and thus adopt the orphaned object.

Note: When you create a new registry object you have no way of specifying the UUID associated with the object; therefore, you cannot simply add a new registry object of the same name to adopt the orphan.

More Detailed Information — Adopting Registry Orphans

The **–uuid** option creates a principal, group, or organization and lets you specify the UUID with which it should be associated instead of assigning it automatically. Except for the way it is created, a principal, group, or organization created by these commands is no different from any other principal, group, or organization. The following illustrates how to use this option to create a principal, group, or organization to adopt an orphaned registry object.

- To create a principal associated the UUID that owns the orphaned object, use the following command:

  ```
  principal create name -uuid uuid [-fullname fullname]
                        [-quota object_creation_quota] [-uid UNIX_number]
  ```

- To create a group associated with the UUID that owns the orphaned object, use the following command:

  ```
  group create name -uuid uuid [-fullname string]
                    [-inprojlist [yes | no]] [-gid UNIX_number]
  ```

- To create an organization associated with the UUID that owns the orphaned object, use the following command:

  ```
  group create name [-fullname string] [-gid UNIX_number]
  ```

  where:

  name Is the primary name for the principal, group, or organization.

  uuid Is the UUID number to be assigned to the principal, group, or organization. This UUID should be the one that owns the orphaned object (that is the one that was associated with the deleted registry object). The UUID is specified in RPC print string format as 8 hexadecimal digits, a hyphen; 4 hexadecimal digits, a hyphen; 4 hexadecimal digits, a hyphen; 4 hexadecimal digits, a hyphen; and 12 hexadecimal digits. The format follows:

  nnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnn

  string Is the full name for the principal, group, or organization.

UNIX_number
For cell principals *only*, is the UNIX number to be associated with the name. If you do not enter this option, the next sequential UNIX number is supplied. For all principals other than cells, the UNIX number is extracted from information that is embedded in the principal's UUID and cannot be specified here.

object_creation_quota
For principals *only*, is the principal's object creation quota. If you do not enter this option, the default is **unlimited**.

–inprojlist For groups *only*, **yes** turns off the project list inclusion so that groups are not included in project lists. If you enter **no**, the group is included in project lists.

Note: In the current implementation of DCE, UNIX numbers are embedded in UUIDs. If you try to create a group or organization to adopt an orphaned object and fail, it could be because the embedded UNIX number is invalid because it does not fall within the range of valid UNIX numbers set for the cell as a registry property. If this is the case, you must reset the range of valid UNIX number to include the UNIX number embedded in the UUID and then try again to adopt the object.

AIX/DCE Security Integration

Introduction — Security Integration

In this release, the AIX base operating security services have been integrated with the DCE security services. This integration is designed to present the typical end user with a single-system image, rather than separate images of a local AIX system and a remote DCE system. There are some limitations to the integration, explained in “More Detailed Information — Security Integration”; however, most users can acquire DCE credentials through AIX commands (such as **login** and **su**), can change their DCE passwords through the AIX **passwd** command, and can retrieve AIX-type information from the DCE registry through the standard AIX **libc.a** routines — **getpwnam()**, **getpwuid()**, **getgrnam()**, and **getgrgid()**. The more general routines, **getpwent()** and **getgrent()**, are not DCE-aware at this time.

More Detailed Information — Security Integration

We strongly recommend that users and groups in the DCE registry and individual system /**etc**/**passwd** and /**etc**/**group** files be synchronized as closely as possible. Synchronization can be facilitated with the **passwd_export** and **passwd_import** utilities after initial cell configuration. Any users who are not synchronized between the cell registry and the local files cannot realize full benefit of the integration feature. On the other hand, this flexible integration scheme supports wandering users (users who are defined in the DCE registry but not in a local system). If a machine is configured to allow it, those wandering users can log onto the system and obtain DCE credentials and local access based on UNIX-relevant information in the registry.

The nonintegrated DCE security command **dce_login** (for logging onto DCE) and the **dcecp** program (for DCE changing passwords and querying the registry) are still available. Of course, local access is prerequisite to using these commands.

The local administrator must set up the system to enable AIX/DCE security integration. This administrator should perform the steps in this section as local user **root**. The system should already be configured as a DCE client machine.

Enabling DCE Access

The **/etc/security/user** file defines attributes for both specific users (those who have a stanza) and other users (those who use the default stanza). To support AIX/DCE security integration, two new user attributes have been defined — **SYSTEM** and **registry**. Use the **SYSTEM** attribute to allow or deny DCE access for particular users or for the system as a whole. Use the **registry** attribute to force password operations either locally or to DCE.

It is important to note that these and other user attributes are applied on a domain-relative name basis. That means that a wandering DCE user who logs onto a system as **jane**, **/.:/jane**, or **/.../this_cell/jane** is affected by any user attributes that can be present in a stanza for a local user **jane** in **/etc/security/user**.

The SYSTEM *Attribute*

The **SYSTEM** attribute describes the method(s) used to authenticate a user to the system. The valid values for this attribute are boolean expression strings made up of the following tokens:

files Authenticate through the local **/etc/passwd** file only.

compat Authenticate through the local files and/or the NIS database.

DCE Authenticate through the DCE registry.

The value for the **SYSTEM** attribute can be a complex expression formed from the methods (tokens) above and the **AND** and **OR** boolean operators. Moreover, the **SYSTEM** expression can specify different results from the methods; the syntax is `method[result]`. Valid results are `UNAVAIL` (the authentication service was unavailable), `NOTFOUND` (the user was not found in the database), `FAILURE` (authentication failed for a different, unspecified reason), or `SUCCESS` (implicit in a method with no result).

The default stanza is created with `SYSTEM = compat`. The local administrator must then change this stanza or a specific user stanza for any user who requires another authentication scheme. The local user **root** should *always* have `SYSTEM = compat` (as well as `registry = files`); in that way, superuser is not dependent on nonlocal authentication mechanisms.

Please refer to the AIX documentation for further discussion of the **SYSTEM** user attribute and its legal values.

The registry *Attribute*

The **registry** attribute defines the database where a user's password is administered. This attribute determines where password queries and changes take place. The valid values for this attribute are:

files The user's password is administered in local files.

NIS The user's password is administered in an NIS database.

DCE The user's password is administered in the DCE registry.

There is no default value for **registry** and, as explained later, the administrator can choose to leave the default undefined. Notice that the local user **root** should always be defined as `registry = files` (as well as `SYSTEM = compat`). In that way, superuser password operations are not dependent on nonlocal mechanisms.

Please refer to the AIX documentation for further discussion of the **registry** user attribute and its legal values.

Protecting Local Resources

The local system administrator must protect any local resources. As stated previously, we highly recommend the DCE registry and local **/etc/passwd** and **/etc/group** files be synchronized as closely as

possible. In some cases, however, it can be desirable not to synchronize a local user with the registry (the local user **root** is a good example). In those cases, the local administrator must see to it that any nonsynchronized names and/or IDs are protected.

The local administrator should take measures to protect any name (user or group name) or ID (UID or GID) known to *not* be synchronized with the DCE registry and that can have data on the local system. For example, if user **joe** exists on the local machine and a different user **joe** exists in the DCE registry, the local administrator should ensure that DCE **joe** cannot gain access to the machine.

The following sections describe various ways to protect local resources.

Protecting Local Resources with the /etc/security/user *File*

Any user who is known not to be synchronized with a user of the same name in the registry should be defined by his **SYSTEM** and **registry** attributes in the **/etc/security/user** file as a local (or NIS) user.

To prevent *any* wandering DCE user from gaining access to the local system, the default stanza should exclude DCE from the **SYSTEM** attribute:

```
default:
        SYSTEM = compat
        registry = files
```

Local users who are synchronized with the DCE registry can still acquire DCE credentials at login by having DCE specified as an authentication method in their individual stanzas.

Protecting Local Resources with the passwd_override *and* group_override *Files*

The standard DCE **passwd_override** functionality is in effect for integrated authentication. That is, the local administrator can prevent DCE authentication from succeeding on the basis of user name, UID, or GID by putting the appropriate entry in the **/opt/dcelocal/etc/passwd_override** file. For example, the following **password_override** entry prevents DCE authentication by user **joe**:

```
joe:OMIT:::::
```

Similar **passwd_override** entries, keyed by UID or GID, can prevent DCE authentication by users having the specified ID.

For base DCE, the **/opt/dcelocal/etc/group_override** file is used for modifying group entries created during a **passwd_export** operation. On a system with AIX/DCE integrated security features, the **group_override** file can also be used to prevent local access by DCE users. To do so, specify a group name and an invalid password in a **group_override** entry. Then, any user who is a member of that group is denied local access. For example, the entry

```
temps:OMIT::
```

prevents any DCE user that belongs to the **temps** group from logging onto the local system. It does not matter whether **temps** is the user's primary group or an arbitrary group in the user's groupset. Access denial by the **group_override** file is in effect only for integrated BOS functions (not DCE-only functions like **dce_login**) and denial is only on a group name basis. To deny access based on GID, use the **passwd_override** file mechanism.

In this release, the **dced** daemon implements the override functionality. Because the override functionality is integral to protecting local resources, DCE authentication is designed to fail if the **dced** daemon is unavailable. The local administrator must ensure that local access is still possible through other authentication methods, if necessary. See the *DCE for AIX Administration Command Reference* for a full discussion of the **passwd_override** and **group_override** files. Also see “Overriding Entries in the Local /etc/passwd and /etc/group Files” on page 7-95 for additional information.

Note: For performance reasons, it is preferable to protect local resources with the **/etc/security/user** file (if possible) rather than the DCE override file.

Configuring DCE Access on a Per-User Basis

The following sections provide specific details about configuring access on a per-user basis.

Configuring and Protecting the Local Root User

Authentication of the local superuser should not depend on any nonlocal mechanisms. Therefore, **/etc/security/user** should always contain a root stanza with the following entries:

```
root:
        SYSTEM = compat
        registry = files
```

In this way, all authentication and password operations are directed toward local files. This also prevents the DCE user **root** from logging onto DCE on this system (unless, oddly, the password for DCE user **root** matches the local root password).

You can also choose to use the **passwd_override** facility to prevent other DCE users with UID **0** from logging onto the system. This should only be necessary if you enable wandering DCE user authentication on your system and if the cell administrator has created an alias for the DCE user **root**. You can determine if an alias exists by viewing the accounts in the DCE registry with **dcecp**.

Configuring and Protecting a Local-Only User

A local-only user is a user who is defined on the local system but is not synchronized with any principal in the DCE registry. Therefore, there can be DCE users whose name or ID collides with the local user. Such a user should have a specific stanza in **/etc/security/user** defining the **SYSTEM** and **registry** attributes as follows:

```
joe:
        SYSTEM = compat
        registry = files
```

Note: If appropriate, **SYSTEM** can be set to **files** and **registry** set to **NIS**. The objective is to expressly exclude DCE from the attributes.

Therefore, local user **joe** does not depend on DCE for authentication or password operations. This also prevents any DCE user **joe** from logging onto DCE on the system.

You can also choose to protect a local-only user's system resources by creating **passwd_override** entries keyed by UID or GID, or by creating **group_override** entries keyed by group name.

Configuring a Synchronized User

In a well-integrated cell, most users are defined and administered in the DCE registry and any user that exists locally should be synchronized with a DCE user. The **passwd_export** utility can be used to synchronize all local user information except for passwords with the DCE registry. The AIX **passwd** user command or the **pwdadm** administrative command should then be used to synchronize passwords.

Any synchronized user can use the AIX BOS commands and have them directed toward the DCE registry. To do this, the user's stanza (whether a specific stanza or the default) in **/etc/security/user** should have a **SYSTEM** attribute that defines DCE as the first authentication method to try, as in the following examples:

```
        SYSTEM = "DCE OR (DCE[UNAVAIL] AND compat)"
```

OR

```
        SYSTEM = "DCE OR (DCE[FAILURE] AND compat)"
```

OR

```
        SYSTEM = "DCE AND compat"
```

Note: For synchronized users, do not set the **registry** attribute.

In the above examples, DCE is the first authentication method tried. If authentication passes, the user is granted access; all UNIX-type information (UID, GID, home directory, login shell) as well as DCE credentials are obtained through DCE. In the first example, if DCE fails because of unavailability, local authentication is attempted. The second example attempts local authentication if DCE authentication fails for any reason. The third example requires both DCE and local authentication to succeed before the user is allowed access to system.

A user's **AUTHSTATE** environment variable is set to the first authentication method that succeeds, and subsequent operations are directed toward the registry defined by **AUTHSTATE** (see “Access Method Identification” in this section).

Do not set the **registry** attribute for synchronized users; the **AUTHSTATE** environment variable is consulted for the proper registry where password operations should be directed. In other words, if **AUTHSTATE** is **DCE**, **registry** is **DCE** and if **AUTHSTATE** is **files** or **compat**, **registry** is **compat**. If the **registry** attribute is explicitly set to a registry that conflicts with the **AUTHSTATE** of the user, password operations can fail.

Supporting Wandering DCE Users

Wandering DCE users are users defined in the DCE registry but not defined on the local machine. You can enable wandering users to log onto DCE from any machine by setting the default **SYSTEM** attribute as follows:

```
default:
        SYSTEM = "DCE OR compat"
```

Any DCE user can log onto DCE from any machine thus configured by supplying his or her DCE name and password. Of course, if local users with colliding names or IDs exist and have been protected either by their own stanzas or by entries in the **passwd_override** or **group_override** files, a wandering user is denied system access.

Access Method Identification

After authentication, an **AUTHSTATE** variable is set in the user's environment. This variable defines the authentication mechanism implemented by the user to gain access to the system and can have the value of **files**, **NIS**, or **DCE**. The **AUTHSTATE** variable determines the first database accessed for a user's subsequent operations. However, if the user's **registry** attribute is defined, password operations are directed toward the database defined by **registry**, regardless of the user's **AUTHSTATE**. The local administrator should make sure that a user's **registry** attribute does not conflict with the user's **AUTHSTATE**, which is determined indirectly by the **SYSTEM** attribute. One method is to leave the **registry** attribute undefined for all but local-only users. Local-only users (such as **root**) should always be defined as `registry = files`.

Changing Passwords

Password operations are directed toward the registry defined by the **registry** user attribute or, in the absence of a **registry** attribute definition, to the registry defined by the **AUTHSTATE** environment variable. Password operations are not directed to both local and DCE registries.

Changing passwords for a DCE-only user is done through **dcecp**, but changing passwords (both DCE and local) for a synchronized user can be done with the AIX **passwd** command in a two-step procedure:

```
$ AUTHSTATE=DCE passwd
$ AUTHSTATE=compat passwd
```

The user should echo the **AUTHSTATE** value before changing it and should set it back to the original value after changing is complete.

Note: Passwords must be kept synchronized for synchronized users. If they are not, either DCE or the local authentication fails. Also, if a user exists locally on more than one machine, the local password must be synchronized on all machines.

Troubleshooting

The following describes some potential situations that can arise, requiring the administrator's attention. Any problems related to a BOS command can also adversely affect the system users as well.

---

dceunixd will not start, issuing the message `(dceunixd) Ensure that valid machine credentials exist and that the KRB5CCNAME environment variable is not set.`

Cause: This can occur if the existing credentials (as determined by the **KRB5CCNAME** environment variable) are nonexistent.

Action: If **KRB5CCNAME** is set in the environment, unset it. Also verify that the machine credentials (**/opt/dcelocal/var/security/cred/dcecred_ffffffff**) exist. You might need to start (or restart) **dced** to create new machine credentials.

---

dceunixd will not start, issuing the message `(dceunixd) Detected another instance of dceunixd, pid` *XXXXX* **(where** *XXXXX* **represents a process ID).**

Cause: This can happen if **dceunixd** is already running on the machine.

Action: Make sure the **dceunixd** process with the stated process ID is running. If it is, a second **dceunixd** cannot be started. If no such process exists, remove the following files:

```
/opt/dcelocal/var/security/adm/dceunixd.skt
/opt/dcelocal/var/security/adm/dceunixd.pid
```

and then reinvoke **dceunixd**.

The dceunixd process grows very large.

Cause: This can happen when a large number of authentication requests take place on a machine. Each DCE authentication causes growth in the authentication process and, because **dceunixd** authenticates on behalf of many clients, it can grow quite large (depending on system activity).

Action: If system performance is degraded because **dceunixd** is consuming a great deal of memory, stop and then restart the daemon. You should consider running a **cron** job where **dceunixd** is stopped and restarted on a regular basis.

An AIX command will not run, issuing the message `Security method "DCE" could not be loaded.`

Cause: This can happen if the machine is not properly configured for AIX/DCE security integration.

Action: Ensure that the local **dceunixd** daemon is running and that the module **/usr/lib/security/DCE** is installed on the machine. It should have the following ownership and permissions:

```
-rwxr-xr-x  1 root    system
```

See "Steps — Security Integration" on page 7-30 for other configuration information.

The ls –l command maps a UNIX ID to the wrong user or group name, or it fails to map an ID to a name.

Cause: This can occur if the **AUTHSTATE** environment variable points to an unexpected database.

Action: Check the **AUTHSTATE** environment variable. All **getpw*** and **getgr*** operations (except **getpwent()** and **getgrent()**) are first directed toward the database indicated by **AUTHSTATE**. That is, if `AUTHSTATE = DCE`, queries go to the DCE registry. If the name or ID is not found in DCE, the local/NIS databases is queried. Alternatively, if `AUTHSTATE = compat`, the local database is queried before DCE. Therefore, the behavior of the query operations can affect the output of ID-to-name mapping commands, such as **ls –l** or **id**. There should be mismapping or ambiguity only on systems where a local user ID collides with a DCE user ID (nonsynchronized users).

Authentication (for example, su, login, telnet, ftp) fails.

Cause: This can happen for a variety of reasons, depending on the system and user configuration.

Action:

1. Check the user's **SYSTEM** attribute. The user (or default) stanza in /**etc**/**security**/**user** must specify the correct authentication mechanism. For a synchronized user (a user defined both locally and in DCE), the **SYSTEM** attribute should be defined so that local-only authentication takes place if DCE authentication fails.
2. Wandering users should try authenticating on a different machine. Access can be denied because of explicit protective measures on the local system.
3. Check that **dced** is running. DCE authentication fails if **dced** is not available.

Registry object not found

Cause: The command may actually have completed successfully.

For example, consider a machine that is configured with security operations being directed to **DCE OR compat** by default (that is, the **SYSTEM** attribute in /**etc**/**security**/**user**'s default stanza equals **DCE OR compat**). If you try to **su** to a local-only user on that machine, the DCE registry is queried in the course of login (since that is the first authentication mechanism specified in the **SYSTEM** attribute). A legitimate "not found" message is returned from DCE and the **su** process displays it before continuing to the next authentication method (**compat**), which should succeed. Issue an **id** command to validate that the authentication took place.

Users should be aware that any given system may be configured so that security operations are directed first to DCE, then to the local system. Consequently, operations (such as **su** and **login**) on local-only users may result in a DCE error message displaying before it actually succeeds in the local case.

Steps — Security Integration

Use the following steps to configure a system for integrated security operations:

1. Ensure that the module /**usr**/**lib**/**security**/**DCE** is installed on the machine.
2. Edit the /**etc**/**security**/**login.cfg** file to include the following lines:

   ```
   DCE:
           program = /usr/lib/security/DCE
   ```

   This defines the authentication method **DCE** to the system.

3. Ensure that the daemon **dceunixd** is running on the machine. This daemon communicates to the DCE servers **secd** and **dced** on behalf of the BOS commands. Refer to the **dceunixd** reference page in the *DCE for AIX Administration Command Reference* for more information.
4. Edit the /**etc**/**security**/**user** stanza file to allow and/or deny DCE access for users. See "Enabling DCE Access" under "More Detailed Information — Security Integration" on page 7-24 for instructions on editing this file.
5. Create or edit the /**opt**/**dcelocal**/**etc**/**passwd_override** and /**opt**/**dcelocal**/**etc**/**group_override** files to explicitly prevent DCE access by certain users. See "Protecting Local Resources" under "More Detailed Information — Security Integration" on page 7-24 for instructions on editing this file.

Restrictions — Security Integration

In this release, there are certain restrictions on the integrated functions and the users that can take advantage of them. These restrictions are:

- Any DCE credentials obtained through the AIX base commands (such as **login**, **su**, or **telnet**) maintain DCE semantics. That is, they expire as determined by DCE registry policy and cannot last for the duration of the local login session. You must be careful when running **rmxcred**; credentials established at login and active in many windows can be deleted and complete login must be performed to regain consistent (cross-window) DCE credentials.
- If you are using DFS, the DFS client should be configured and running on your machine before **dceunixd** is started. Any user that instigates integrated **login** or **dce_login** to obtain DCE credentials before DFS is running will not have authenticated access to DFS files.
- Any DCE user with a cell-relative name longer than eight characters cannot take advantage of the integration feature. The eight-character limit is UNIX-imposed.
- Any DCE user with a cell-relative name containing a slash (/) cannot take advantage of the integration feature. Therefore, the DCE user **mary** can take advantage of integrated features; the DCE user **writers**/**mary** cannot.
- Only local cell users are supported; foreign principals cannot use integrated operations. Foreign principals can still use **dce_login** and **dcecp** for intercell access.
- Certain AIX-generated data for local users can be misrepresented on systems that allow wandering DCE user login and where local users have not been sufficiently protected. This data includes last login time and information in the system audit trails. Information for a local user is affected if the system is accessed by a DCE user having the same name. For example, consider a scenario where a local system defines a user **joe** and this user differs from a DCE user **joe**. Assume that there are no collisions between UIDs or GIDs on the local system and the DCE registry. The local administrator has failed to override login by DCE user **joe** (through the **passwd_override** file) and has enabled wandering user login by setting the default **SYSTEM** value in /**etc**/**security**/**user** to be `compat OR (compat[FAILURE] AND DCE)`.

  If DCE user **joe** logs into this system, he gains access and has the UID and GID as obtained from the DCE registry. Therefore, DCE **joe** cannot access local **joe**'s files because they are owned by local **joe**'s UID and GID. However, the local system still knows the user as **joe** and it cannot distinguish the domain to which he belongs (DCE or local) in some circumstances. Therefore, data logged is attributed to **joe** regardless of whether DCE **joe** or local **joe** generated the data.

  The administrator should guard against this problem by synchronizing all local users possible with the DCE registry, and by adequately protecting any unsynchronized users (by applying local overrides for colliding names and IDs and by configuring the /**etc**/**security**/**user** stanzas properly).
- The local **root** user cannot **su** to a DCE user and obtain DCE credentials. This is because the local **root** user is not prompted for a password, which is necessary for DCE login. In fact, any command that does not prompt for a password (for example, **rcp**, **rsh**, or **rlogin** when an **.rhosts** file is present)

cannot be used to acquire DCE credentials. BOS commands acquiring DCE credentials are *DCE-aware* in that they use the DCE registry for user and password information. They do not use the underlying Kerberos authentication protocol.

- The AIX **passwd** command changes the DCE password only for those DCE users whose **pwd_val_type** extended registry attribute specifies a user-selected password. It cannot change the password for those DCE users whose **pwd_val_type** ERA specifies a system-generated password.
- The local DCE registry is not sufficient to pass DCE authentication. If the DCE security server is unavailable and DCE is defined as a required authentication method by the **SYSTEM** attribute, the user cannot access the system.
- Password operations are directed toward either the DCE registry or the local files, not both. Therefore, changing the password for a user who exists in both DCE and on a local system (as a synchronized user) is a two-step process. You can use the AIX **passwd** command to first change the password in one registry and then in another. For example:

```
$ AUTHSTATE=DCE passwd
$ AUTHSTATE=compat passwd
```

Some AIX BOS commands that rely on **/etc/passwd** or **/etc/group** information are not DCE-aware. These commands include the **finger**, **quot**, **lsuser**, and some administrative commands (such as **mkuser**, **rmuser**, **mkgroup**, and **rmgroup**).

Examples — Security Integration

- User **joe** (whose local **/etc/passwd** information is synchronized with the DCE registry) can have a stanza configured so that he obtains DCE credentials and local access at login. His stanza can look something like this:

```
joe:
       SYSTEM = "(DCE AND compat) OR (DCE[UNAVAIL] AND compat)"
```

  This indicates that **joe** must pass both DCE and **compat** authentication *or* he must fail DCE authentication due to DCE unavailabilty and pass **compat** authentication to successfully log onto the system.

- As another example, wandering DCE user **bob** can try to log onto a system (on which he is not defined locally) where the default stanza contains the following:

```
default:
        SYSTEM = "DCE OR (DCE[UNAVAIL] AND compat)"
```

  This indicates that a default user must pass DCE authentication *or* fail DCE authentication due to DCE unavailabilty and pass **compat** authentication to gain access to the system. Of course, if the user is a wandering user and has no local entry in **/etc/passwd**, the user cannot pass **compat** and is denied any access in the event DCE is unavailable.

- The stanza for local user **joe** would include the following entries if he is known not to be synchronized with a user of the same name in the registry:

```
joe:
        SYSTEM = compat
        registry = files
```

  The stanza for NIS user **jane** would include the following entries:

```
jane:
        SYSTEM = compat
        registry = NIS
```

  The objective is to expressly exclude DCE from the attributes. If DCE **joe** or **jane** wanders up to the machine, the only way they can log on is to provide the password that makes **compat** (local or NIS) authentication succeed.

- If user **jane** accesses the local system through DCE, her **AUTHSTATE** variable is set to **DCE**. If she then executes an **ls –l** command on some file, the DCE registry is queried to map the UID and GID to names. If the IDs do not exist in the DCE registry, the **ID query** operation falls back to **compat**, and the local /**etc**/**passwd** and /**etc**/**group** files are queried. Users should be aware that IDs can be mismapped, depending on the database preferentially queried and the file system being accessed (DFS or local).

Authentication Policy

Introduction — Authentication Policy

Authentication policy regulates ticket lifetimes. You can set authentication policy for the registry as a whole, using the **dcecp registry modify** command and for individual accounts using the **dcecp account modify** command. The authentication policies you can set are described in the following sections.

Note: Be aware that, in addition to the authentication policies described in this section, you can also control preauthentication policy for a principal by attaching an instance of the *pre_auth_req* ERA to the principal. See “Creating and Maintaining Principals, Groups, and Organizations” on page 1-48 for a general discussion of preauthentication and information on preauthentication administration.

More Detailed Information — Authentication Policy

Maximum Ticket Renewable Lifetime

Note: This feature is not currently used by the DCE; any use of this option is unsupported at the present time.

The maximum ticket renewable time (**maxtktrenew** attribute) that you set determines the maximum amount of time in hours before a principal's ticket-granting ticket expires and the time the principal must log in again to reauthenticate and obtain another ticket-granting ticket. The shorter you make the maximum ticket renewable lifetime, the greater the security of the system. However, since users must log in again to renew their ticket-granting ticket, the time needs to take into consideration user convenience and the level of security that your cell requires.

You define maximum ticket renewable time with the **dcecp maxtktrenew** attribute in the form:

maxtktrenew *hours*

hours is a number that indicates the number of hours before a principal's ticket-granting ticket expires.

Note that can set this time for individual accounts by using the **account modify** command.

Maximum Ticket Lifetime

The maximum ticket lifetime (**maxtktlife** attribute) is the maximum amount of time in hours that a ticket issued to a principal is valid. When a client presents a ticket to a server, the lifetime that is granted to the ticket takes into account the maximum ticket lifetime that is set for both the server and the client. The lifetime that is granted will not exceed the shorter of the server's and client's maximum ticket lifetime.

You define maximum ticket lifetime with the **dcecp maxtktlife** attribute in the form:

maxtktlife *hours*

hours is a number that indicates the number of hours that a ticket issued to a principal is valid.

The shorter you make the maximum ticket lifetime, the greater the security of the system. However, extremely frequent renewal can cause processing overhead. The maximum ticket lifetime that you set needs to take into consideration system performance and the level of security that you require.

Note that can set this time for individual accounts by using the **account modify** command.

Handling Conflicting Policies

Different standard and authentication policies can be in effect for the registry as a whole and for individual organizations (for standard policy) and accounts (for authentication policy). If the policy that is set for the registry as a whole differs from the policy that is set for an individual organization or account, the stricter policy applies. For example, suppose registry policy specifies a minimum password length of six characters and policy for the organization named **classic** specifies eight characters. If you create the account **bach cantata classic**, the stricter policy (in this case, the organization policy) applies, and the account password must be at least eight characters long. Table 7-1 lists the stricter policy for each policy type.

Table 7-1. Stricter Standard Policies

For This Type of Policy...	This is the Stricter Policy...
Password expiration date	The shorter expiration period.
Password lifespan	The shorter lifespan.
Account lifespan	The shorter lifespan.
Password length	The greater length.
Password consisting of all spaces	The password cannot consist entirely of spaces. It must include characters.
Password consisting of all alphanumeric characters.	The password cannot consist of all alphanumeric characters. It must include some nonalphanumeric characters.
Maximum ticket renewable	The shorter time. This feature is not currently used by the DCE; any use of this option is unsupported at the present time.
Maximum ticket lifetime	The shorter time.

When the registry is created, standard policies are by default at their most permissive state; that is, the password expiration date is **none**, password and account lifespans are **unlimited**, the minimum password length is **0**, and passwords can consist of all spaces and all alphanumerics. The maximum ticket lifetime is set to 10 hours. (Maximum ticket renewable is not currently used.) To implement stricter policies, you must use the **registry modify** command.

The Effects of Changes on Existing Policies

Except for the password format policies described in “Password Format” under “More Detailed Information — Policies” on page 7-104, policy changes affect all existing accounts and all accounts that you create after the change.

Changes to password format policies, such as password length, whether passwords can consist of all spaces, and whether passwords can consist of all alphanumeric characters, affect only passwords for those accounts that are created after the policy is changed. The changes have no effect on existing passwords. For example, if you change the minimum password length policy to enforce a longer length password, existing passwords that are shorter than the length specified by the new policy are unaffected. They do not need to be changed, but any new passwords that are created must adhere to the new policy. However, the next time you change an existing password, the longer length policy is enforced.

Steps — Authentication Policy

To display policy:

- For the registry as a whole, use the **dcecp registry show** command with the **–policies** option.
- For an individual organization or account, use the **dcecp organization show** command with the **–policies** option (for standard policies) or the **dcecp account show** command with the **–policies** option (for authentication policies).

To set policy:

- For the registry as a whole, use the **dcecp registry modify** command. The following sample command uses the **pwdlife** option to set the password lifespan policy for the registry as a whole to 180 days:

  ```
  dcecp> registry modify -pwdlife 180
  ```

- For an individual organization or account, use the **dcecp organization modify** command for standard policies or the **dcecp account modify** command for authentication policies. The following sample command uses the **–pwdlife** attribute option to set the password lifespan policy for the organization **classic** to **unlimited**:

  ```
  dcecp> organization modify classic -pwdlife unlimited
  ```

Note that the commands shown above all use attribute options. You can also set policy by using the **dcecp registry modify**, **dcecp account modify**, and **dcecp organization modify** commands with the **–change** option and attribute lists. For example, to use an attribute list to set the password lifespan policy for the organization **classic** to **unlimited**, the command would be:

```
dcecp> organization modify classic -change {pwdlife unlimited}
```

Automatic Changes to the Locksmith Account

If the locksmith account exists when you start the Security server in locksmith mode, the Security server checks certain account and registry policy information and makes the changes as shown in Table 7-2 and Table 7-3 on page 7-36. These changes ensure that, even if the account or registry policy was tampered with, you will now be able to log into the locksmith account. For example, if an intruder changes the Account Lifespan registry policy to 1 minute, the locksmith account will never be valid long enough to be used. Therefore, if the Security server finds that the Account Lifespan registry policy is set to less than what is required for the locksmith account to be valid for at least 1 hour, it changes the Account Lifespan policy to be the time difference between the creation time of the locksmith account and the time 1 hour from the current time.

Table 7-2. Locksmith Account Changes Made by the Security Server

If the Security server finds the ...	**It changes the ...**
Password-Valid Flag is set to no	Password-Valid Flag to yes
Account Expiration Date is set to less than the current time plus 1 hour	Account Expiration Date to the current time plus 1 hour.
Client Flag is set to no	Client Flag to yes
Account-Valid Flag is set to no	Account-Valid Flag to yes
Good Since Date is set to greater than the current time	Good Since Date to the current time
Password Expiration Date is set to less than current time plus 1 hour	Password Expiration Date to the current time plus 1 hour

Table 7-3. Registry Policy Changes Made by the Security Server	
If the Security server finds the ...	**It changes the ...**
Account Lifespan is set to less than the difference between the locksmith account creation date and the current time plus 1 hour	Account Lifespan to the current time plus 1 hour minus the locksmith account creation date
Password Expiration Date is set to greater than the time the password was last changed, but less than the current time plus 1 hour	Password Expiration Date to the current time plus 1 hour

Backing Up and Restoring the Registry Database

Introduction — Backing Up and Restoring the Registry Database

You must use the exact procedures that are described here to back up the registry database to prevent backups from arriving at the master replica during the backup.

Only the master replica database and its master key file need to be backed up. Use the procedures described in the following sections when you back up the entire disk on which the master replica and its master key are stored, and when you back up only the master's database files and its master key file.

Steps — Backing Up and Restoring the Registry Database

Backing Up the Registry Database

To run the backup procedures, ensure that you are logged into the DCE with an administrative account. Run the DCE control program to perform the backup, using the following steps:

1. Enter the **registry disable** command to set the master replica to the maintenance state. The following command sets the primary registry in the cell **giverny.com** to maintenance state:

   ```
   dcecp> registry disable /.../giverny.com/subsys/dce/sec/master
   dcecp>
   ```

   Setting the master replica to the maintenance state causes the master to save its database to disk and refuse all updates.

2. Back up the primary registry by backing up either the entire volume or the **/opt/dcelocal/var/security/rgy_data** tree (the registry) and the **/opt/dcelocal/var/security/.mkey** file, which is the file that contains the master key used to encrypt all keys in the registry. Note that, because the **/opt/dcelocal/var/security/.mkey** file contains the master key, restoring a backup of the registry database is useless unless the **/opt/dcelocal/var/security/.mkey** file is also restored.

   The exact commands that are used for the backup are a matter of personal preference. However, if you write both the database and the master key file to the same tape, store the tape in a locked area with restricted access. Alternatively, you can write the database and the key file to separate tapes and store each tape in a different location.

3. When the backup completes, take the master replica out of maintenance state, as follows:

   ```
   dcecp> registry enable  /.../giverny.com/subsys/dce/sec/master
   dcecp>
   ```

   The security server resumes accepting updates.

Note that the previous examples supplied the name of the registry primary site to the **registry enable**. and **registry disable** commands. If you do not supply a registry site name, the commands use the site named

in the **_s(sec_)** variable. If this variable is not set, the commands use the primary registry of the machine's default cell. See "Setting the **_s(sec)** Variable" on page 7-138 for more information.

Restoring the Registry Database

This section provides instructions for restoring the master replica's database files and master key file. The procedure assumes that the database is being restored to the same machine from which it was backed up, and that you are using the DCE control program. If you are moving the database to a different machine, follow the instructions in "Changing the Master Replica Site."

To restore the registry database to a machine, perform the following steps:

1. Log in as **root** at the primary registry site.
2. If **secd** is running, stop it by issuing the **registry stop** command. When you use this command, you must supply the fully qualified name of a specific replica as an argument. The following sample command stops the **secd** named **master**.

   ```
   dcecp> registry stop /.../giverny.com/subsys/dce/sec/master
   ```

3. Copy the backup files from the backup media to the machine. If you have backed up only the registry data files and the master key files, be sure to copy the registry database to the **/opt/dcelocal/var/security/rgy_data** tree and the master key file to **/opt/dcelocal/var/security/.mkey**. Note that, because the **/opt/dcelocal/var/security/.mkey** file contains the master key, restoring a backup of the registry database is useless unless the **/opt/dcelocal/var/security/.mkey** file is also restored.
4. Restart the server by invoking **secd** with the **–restore_master** option, as follows:

   ```
   $ /opt/dcelocal/bin/secd -restore_master &
   ```

   This command will start **secd** and cause the master replica to mark all slave replicas to be reinitialized.
5. Verify that **secd** starts automatically at system startup.

Note: If you are restoring *only* a master key file and have not changed the master key, you can simply copy the master key file from the backup media without performing all of the other steps that are in the restore procedures.

Changing the Master Replica Site

Introduction — Changing the Master Replica Site

The machine that runs the master replica server must be available at all times. If you are planning to remove this machine from your network or to shut it down for an extended period, you need to change the site of the master replica.

The preferred method for changing the primary registry site is to use the **dcecp registry designate** command to reverse the roles of the primary server and a secondary server. In other words, make the primary the secondary and the secondary the primary.

More Detailed Information — Changing the Master Replica Site

When you invoke the **dcecp registry designate** command, the following occurs:

1. The current master sends all pending updates and its propagation queue to the replica designated as the new master.
2. The designated new master reads the current master's replica list to obtain information required for it to manage propagation to the secondaries.
3. When the designated new master has obtained all necessary information from the current master, it becomes the new master, and the current master becomes a slave.

Because this orderly and complete transfer of information ensures no data is lost, the **dcecp registry designate** command is the preferred method to move the primary registry to another machine when the registry servers at the primary and secondary sites are operating normally. Note that the **dcecp registry designate –master** command is also available to change a replica from a slave to the master. However, because the **dcecp registry designate –master** command can cause data to be lost, use it *only* when the current master has been destroyed. It is not recommended in instances when the master is unreachable because of a network failure or because the master has gone down temporarily. See "Troubleshooting Procedures" on page 1-58 for more information on the **dcecp registry designate –master** command.

Steps — Changing the Master Replica Site

Follow these steps to change the site of a master replica:

1. Choose the new master site. A slave replica must exist at this site. If necessary, use the configuration process to configure a secondary machine. See *DSS for AIX Up and Running!* for configuration information.
2. Issue the **registry designate** command to set default replica to the current master replica. In the example below the master replica is set the replica named **master** in the cell **giverny.com**:

   ```
   dcecp> registry designate _s(sec) /.../giverny.com/subsys/dce/sec/master
   dcecp>
   ```

3. Issue the **registry designate** command to reverse the roles of the master and slave replicas. When you issue this command, you supply the name of the replica to be made the new master as an argument. The following example makes the replica named /**.../giverny.com/subsys/dce/sec/music** the new master:

   ```
   dcecp> registry designate /.../giverny.com/subsys/dce/sec/music
   dcecp>
   ```

4. Verify that the master site changed by issuing the **registry catalog** command.

Changing the Registry Master Key

All passwords stored in the primary registry are encrypted by a master key. Note that the master key is created when you create the registry database during system configuration.

You can use the **dcecp registry modify** command with the **–key** option to change the registry's master key and to re-encrypt all passwords with the new master key. Each replica (primary and secondary) maintains its own master key to access the data in its copy of the registry.

You should change each replica's master key on a regular basis. Before you run either program to do this, ensure that you are logged into an administrative account.

The following command line changes the master key and re-encrypts all passwords for the replica **art_server_1**.

```
registry modify /.../giverny.com/subsys/dce/sec/art_server_1 -key
```

Container ACLs

Introduction — Container ACLs

The Object ACL controls access to the object itself. A container object has, in addition to its Object ACL, an Initial Container ACL and an Initial Object ACL. These two ACLs are not used for access control as such, but instead for cloning initial ACLs for objects or containers created within the initial container. The Initial Container ACLs and the Initial Object ACLs can be edited in the same way as the usual ACL, by using the **–ic** and **–io** options to the **dcecp acl** commands.

More Detailed Information — Container ACLs

Objects and Containers

The type of ACL used for an object depends on whether the object is a simple object or a container. Containers are objects that hold other objects. The objects they hold can themselves be either simple objects or container objects. Simple objects do not hold other objects. Although any DCE component can have objects and containers, the simplest and most common illustration is the file system. In the file system, there are files and directories. The files are simple objects, and the directories are containers. The directories can hold simple objects (files) and other containers (subdirectories).

A simple object only has an object ACL. A container object has an object, an Initial Container, and an Inital Object ACL.

Initial ACLs for Objects and Containers

Initial ACL entries and the ACL that contains them are applied automatically when an object is created. The entries can be modified at any time with the DCE control program. The types of DCE ACLs used as Initial ACLs for containers and objects are as follows:

- The Initial Container ACL determines the default object ACL and an Initial Container ACL for containers created within a container. For example, the file system Initial Container ACL for a directory specifies the default object ACL and Initial Container ACL for subdirectories created within that directory.
- The Initial Object ACL determines the default object ACL for simple objects created within a container. It determines the default Initial Object ACL for containers created within a container. For example, the file system Initial Object ACL for a directory specifies the default object ACL for files created within that directory.

Default ACLs for Objects

When a simple object is created in a container, it inherits the container's Initial Object ACL as its Object ACL. The following figure illustrates how the default ACL is assigned to simple objects created in containers.

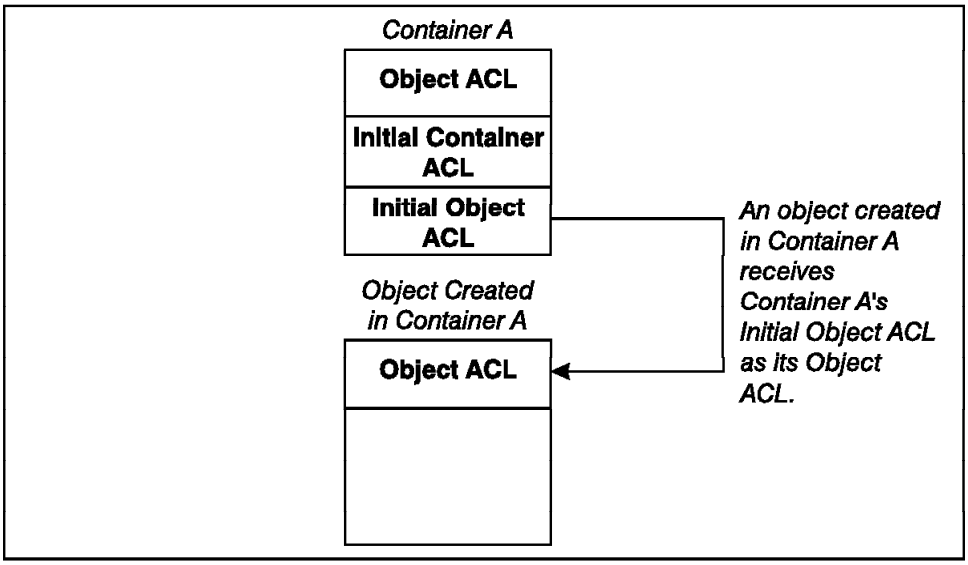


Figure 7-3. Initial ACLs for Objects Created in Containers

Default ACLs for Containers

When a container is created within a container (a subdirectory within a directory, for example), it inherits the parent container's

- Initial Container ACL as its Object ACL and as its Initial Container ACL, and
- Initial Object ACL as its Initial Object ACL.

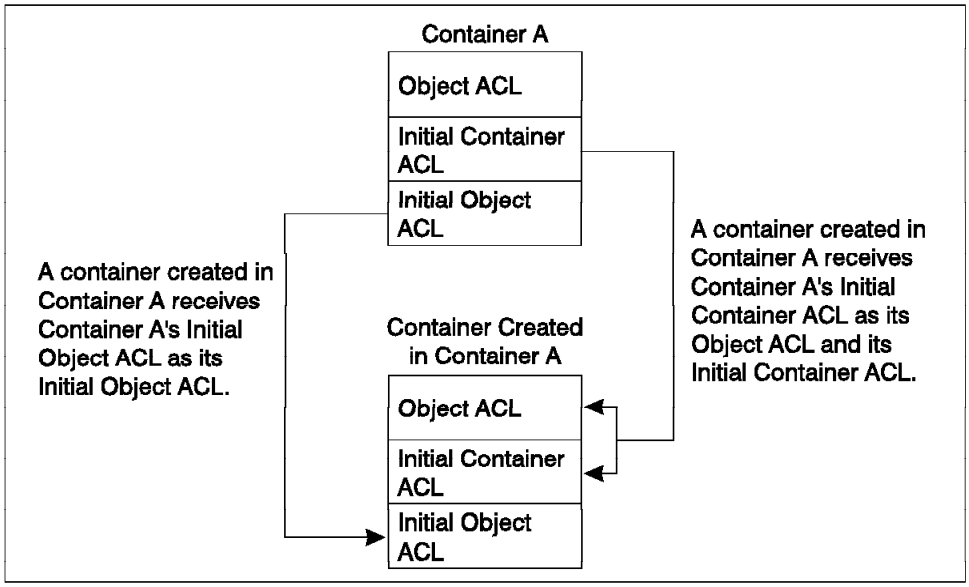


Figure 7-4. Initial ACLs for Containers Created in Containers

Effect of Masks When Editing ACLs

If the user specifies a new **mask_obj** ACL entry, **acl modify** uses it. Otherwise, the **acl modify** command recalculates the mask, using the algorithm shown in the following paragraph, unless the user has specified one of the **–mask calc**, **–mask nocalc**, or **–purge** options. Therefore the mask can change, granting more or fewer permissions, on every **acl modify** command.

Here is the algorithm that the **acl modify** command uses when calculating the mask:

1. Retrieve the existing ACL of the file.
2. Perform all requests to remove entries and to reduce the permissions of existing entries.
3. Calculate the union of the actual permissions of all remaining entries.
4. Determine which permissions differ between the actual and effective rights. (This is the logical XOR of the results of Steps 3 and 4.)
5. Perform all requests to add new entries to the ACL and all requests to increase the permissions of existing entries.
6. Calculate the union of these newly granted permissions and the old effective permissions (from Step 4). This is the candidate new mask value.
7. If there are any permissions in the candidate new mask that are also in the permissions that differ between the original actual and effective rights (from Step 5), applying the candidate new mask would unexpectedly grant some new right that the user did not intend. Unless the user specified one of the options **–mask calc**, **–mask nocalc**, or **–purge**, this condition is an error, and the ACL is not modified. Otherwise, the candidate new mask is applied as the new mask.

For the vast majority of ACL operations, such automatic recalculation is safe. In certain rare cases, the recalculation of the mask can grant additional rights that the user did not expect; for instance, a permission granted to an entry that the user did not specify and that was not among the entry's previous effective rights.

Examples — Container ACLs

- If you create a file named **report** in the directory **marketing**, the system assigns **report** the Initial Object ACL of the directory **marketing**. If you create a subdirectory in **marketing**, the system assigns the new subdirectory the Initial Container ACL of **marketing**. New subdirectories also receive a set of initial ACLs that match the parent directory's initial ACLs. In this example, the new subdirectory also receives **marketing**'s initial ACLs as its own ACLs.
- The following example shows how ACLs are initially assigned to containers created within containers.

  Assume Container A has the following ACLs:

  Object ACL

  ```
  {user_obj crwxid}
  {group_obj crwxid}
  {other_obj r}
  ```

  Initial Container ACL

  ```
  {user_obj crwxid}
  {group_obj rw}
  {other_obj r}
  ```

  Initial Object ACL

  ```
  {user_obj crwxid}
  {group_obj r}
  {other_obj r}
  ```

  When Container B is created in Container A, it has the following default ACLs:

  Object ACL (Container A's Initial Container ACL)

  ```
  {user_obj crwxid}
  {group_obj rw}
  {other_obj r}
  ```

  Initial Container ACL (Container A's Initial Container ACL)

```
{user_obj crwxid}
{group_obj rw}
{other_obj r}
```

Initial Object ACL (Container A's Initial Object ACL)

```
{user_obj crwxid}
{group_obj r}
{other_obj r}
```

- The following example shows how mask recalculation works and also shows the effect of the options.

  Observe that the ACL contains an entry granting **rwx** permission to some user, but the mask allows an effective permission of **r-x**. Adding a new **rwx** ACL entry and recalculating the mask (according to Step 6) to **rwx** is unsafe because the first user's effective access rights are unexpectedly changed from **r-x** to **rwx**. If the **acl modify** command detects such an unsafe condition, its default action is to issue an error message and not change the ACL.

  The initial state, showing the permissions and the effective permissions, is:

  ```
  dcecp> acl show /.:/concertos
  {user vivaldi rwx effective r-x}
  {mask_obj r-x}
  ```

  Adding a user as shown results in an error because the mask recalculation would give **vivaldi** an effective permission of **rwx**:

  ```
  dcecp> acl modify /.:/concertos -add {user telemann rwx}
  Error: Unintended permissions not granted.
  ```

  Using the **–mask calc** option explicitly allows the recalculated mask to be applied in spite of the new permission granted to **vivaldi**. The mask is set to the union of the permissions granted to the file group class entries on the ACL. This option can result in the inadvertent granting of extra permissions.

  ```
  dcecp> acl modify /.:/concertos -add {user telemann rwx} -mask calc
  dcecp> acl show /.:/concertos
  {user vivaldi rwx effective rwx}
  {user telemann rwx effective rwx}
  {mask_obj rwx}
  ```

  Using the **–mask nocalc** option explicitly retains the **r-x** mask, resulting in reduced effective permissions for **telemann**. The ACL is modified exactly as specified by the user, and no mask calculation or purging of permissions occurs.

  ```
  dcecp> acl modify /.:/concertos -add {user telemann rwx} -mask nocalc
  dcecp> acl show /.:/concertos
  {user vivaldi rwx effective r-x}
  {user telemann rwx effective r-x}
  {mask_obj r-x}
  ```

  Using the **–purge** option replaces the actual permissions with the effective permissions in all entries. More precisely, if the command detects an unsafe condition, then the condition intersects the current value of the mask with all of the existing, unmodified entries in the file group class, replacing all ACL entries (except **user_obj**, **other_obj**, **mask_obj** and **unauthenticated**) with their effective permissions.

  ```
  dcecp> acl modify /.:/concertos -add {user telemann rwx} -purge
  dcecp> acl show /.:/concertos
  {user vivaldi rwx effective r-x}
  {user telemann rwx effective rwx}
  {mask_obj rwx}
  ```

Controlling Access to the Audit Daemon

Introduction — Controlling Access to the Audit Daemon

You must control access to the Audit daemon to prevent unauthorized application servers (the Audit clients) from using it. If an unauthorized server is able to log its audit records, the Audit storage space would be exhausted.

You control access to the Audit daemon by editing the Access Control List (ACL) of the Audit daemon object, /**.:/hosts**/*hostname*/**audit-server**, using the DCE control program.

More Detailed Information — Controlling Access to the Audit Daemon

DCE Permissions Supported by the DCE Audit Service

The DCE Audit Service supports the following DCE permissions that can be used to define the ACL of the Audit daemon:

r Read permission. Allows a principal to read the filters.

w Write permission. Allows a principal to modify the filters.

c Control permission. Allows a principal to control the Audit daemon. This includes the ability to enable or disable the logging service, and to modify the ACL of the Audit daemon.

l Log permission. Allows a principal to write audit records in the audit trail file.

Initial ACL of the Audit Daemon

The initial ACL of a host's Audit daemon contains the following entries:

```
{unauthenticated -r--}
{user hosts/nodoz/self crwl}
{group subsys/dce/audit-admin crwl}
{any_other -r--}
```

The first entry allows any unauthenticated user only read access to the filters. The second entry allows the host principal (**hosts**/*hostname*/**self**) to query and modify the filters, control the Audit daemon, and to write to the audit trail file. The third entry allows the members of the group **subsys**/**dce**/**audit-admin** the same access rights as the host principal. The last entry allows all other principals, only read access to the filters. You can modify this ACL to suit your security requirements using the DCE control program.

Giving Permissions to Audit Clients and Administrators

Using **dcecp**, you can add entries to the ACL of the Audit daemon that will grant audit clients the log permission to the audit trail file.

All Audit clients can then be made members of this group and inherit its permissions to the Audit daemon.

ACL entries must also be added to grant designated administrators the read, query, and control permissions to the Audit daemon.

Examples — Controlling Access to the Audit Daemon

- You can create a DCE Security group that consists of the servers on the host that are authorized to generate audit records (for example, **group**/**hosts**/*hostname*/**audit-clients**). Give this group the log permission to the Audit daemon. For example:

```
dcecp> acl modify /.:/hosts/machine1/audit-server -add
                  {group hosts/machine1/audit-clients l}
```

- For the administrator's group **group/hosts/machine1/audit-admin**, enter:

```
dcecp> acl modify /.:/hosts/machine1/audit-server -add
                  {group hosts/machine1/audit-admin rwc}
```

Controlling and Displaying Audit Trails

Introduction — Controlling and Displaying Audit Trails

Audit daemons log audit records sent from audit clients into an audit trail file. If the Audit daemon is started without any argument, then the audit trail file used is **/opt/dcelocal/var/audit/adm/central_trail**. You can also direct the audit trail to another file by using the **–t** option of when starting daemon and specifying the path name of the file to which the logs should be written.

More Detailed Information — Controlling and Displaying Audit Trails

Displaying Audit Trail Files

Use the DCE control program's **auditrail show** command to examine the contents of an audit trail file. You can display the contents of either the central or local audit trail file.

If you prefer to have the audit trail data put into a file instead of displayed on your screen, include the **–to** option in the **audtrail show** command line. This option prints the audit trail file's contents to a specified file name. Using this option is strongly recommended for large trail files.

Controlling the Audit Trail Size

By default, audit trail files are limited to a size of 2 MB. When the audit service detects that the trail file will grow larger than this value, it closes the file, creates a new unique name for the file using timestamp information, and then opens a new trail file with the original name. It then proceeds to write new audit logs to this file. When this file grows too large, this process is repeated.

To change the size of the audit trail file, set the environment variable **DCEAUDITTRAILSIZE** to the size you require before starting the application that is using the Audit Service. Setting this environment variable overrides the default 2 MB size limit.

You can also allow the Audit daemon to wrap around the central trail file when its limit (the default 2 MB or set by **DCEAUDITTRAILSIZE**) is reached. To do this, you should start the Audit daemon with the **–wrap** option:

```
$ auditd -wrap
```

You may also want to use this option if old audit records have little or no value, and you only want to keep relatively recent records.

A trail size limit can also be set using the **–s** option of the **auditd** command. The limit set using this method overrides the default 2 MB limit.

If for any reason you want to take a snapshot of the audit trail before it reaches the limit, you can use the DCE control program's **aud disable** command to disable logging and then copy the file. You can then use the DCE control program's **aud rewind** command to rewind the central audit trail file. (Note that if required, you can back up this audit file at this time. But, if backup is desired, it is best to let the Audit Service automatically create new trail files and back these up.) Then use the **aud enable** command to enable the Audit daemon's logging service again.

Changing the Audit Trail File Storage Option

The storage strategy option can be changed while the Audit daemon is running. This can only be performed on the central audit trail file.

Examples — Controlling and Displaying Audit Trails

- You can use the following command to see the contents of the audit trail file **central_trail**.

```
dcecp> audtrail show  /opt/dcelocal/var/audit/adm/central_trail

--- Start of an event record --- Event Number: 259
Client: /.../stp.gburg.ibm.com/hosts/drinkernisti/self
Event Outcome: success
Authorization Status: Authorized with a pac
Local Time: 1994-12-19-19:02:27.037-05:00I-----
--- End of an event record ---
--- Start of an event record --- Event Number: 256
Client: /.../stp.gburg.ibm.com/hosts/drinkernisti/self
Event Outcome: success
Authorization Status: Authorized with a pac
Local Time: 1994-12-19-19:02:28.819-05:00I-----
--- End of an event record ---
```

- To use a trail file size of 5MB, set the value of **DCEAUDITTRAILSIZE** as follows:

```
DCEAUDITTRAILSIZE 5000000
```

- The following example shows how the **aud modify** command is used to cause the audit trail to wrap when it reaches the limit of the file:

```
dcecp> aud modify -stostrategy wrap
```

  This example command changes the value of the Audit daemon's storage strategy attribute to **wrap**.

Copying ACLs

Introduction — Copying ACLs

To copy an ACL from one DCE object to another, use the DCE control program **acl replace** command with the **–acl** option as shown here:

```
dcecp> acl replace /.:/hosts/hermes -acl [acl show /.:/hosts/cyclops]
```

This command replaces the ACL for the host **hermes** with the ACL for the host **cyclops** whose name is specified in the **acl show** command invoked by the **–acl** option. Note how the **–acl show** command in the **–acl** option is enclosed in brackets ([and]). This is required when the **–acl** option value is a command invocation.

More Detailed Information — Copying ACLs

If you are copying between cells, use the **acl replace** command's **–cell** option, as well as its **–acl** option. For example:

```
dcecp> acl replace /.:/hosts/hermes -acl
                   [acl show /.:/hosts/cyclops]
                   -cell [acl show /.:/hosts/cyclops -cell]
```

To copy an **extended** entry type from the domain of one ACL Manager to the domain of another ACL Manager, use the output from the **dcecp** program's **acl show** command as the input to an **acl replace**

command. To copy **extended** entries this way, both ACL Managers must support the **extended** entry type.

Creating and Maintaining Aliases for Principals and Groups

Introduction — Creating/Maintaining Principal/Group Aliases

Use the **dcecp principal create** command to create and maintain aliases for principals and groups. Organizations cannot be given aliases.

More Detailed Information — Creating/Maintaining Principal/Group Aliases

Creating Aliases

To create an alias for a principal, enter the **dcecp principal create** command in the following form:

principal create *name* **-uid** *unix_ID* **-alias**

where:

name Is the alias name for the principal or group.

unix_ID Is the UNIX ID that is associated with the principal for which you are creating the alias.

–alias Indicates the *name* is an alias.

To create an alias for a group, enter the **dcecp group create** command in the following form:

group create *name* **-gid** *groupunix_ID* **-alias**

where:

name Is the alias name for the principal or group.

groupunix_ID Is the UNIX ID that is associated with the group for which you are creating the alias.

–alias Indicates that *name* is an alias.

Changing Primary Names to Aliases and Vice Versa

To change an alias to a primary name or a primary name to an alias, use the **dcecp principal modify** command for a principal or the **dcecp group modify** command for a group. These commands have the following form:

principal modify *name* **-alias {yes|no}**

group modify *name* **-alias {yes|no}**

where:

name Is the primary name to be changed to an alias or the alias to be changed to a primary name.

–alias **–alias yes** changes the primary name identified by *name* to an alias. **–alias no** changes the alias identified by *name* to the primary name.

A principal or group can have only one primary name at a time. Before you change an alias to a primary name, first change the primary name to an alias.

Creating and Maintaining Attribute Instances

Introduction — Creating and Maintaining Attribute Instances

Using **dcecp** operations, you can attach extended registry attributes to objects, modify the values assigned to those attributes, and delete the attachment in the same manner as you would any attribute attached to an object.

You can attach extended registry attributes to any of the following registry objects using the **dcecp create** and **modify** commands:

- principal
- group
- organization
- policy (In the current release, you cannot attach attributes to this object.)
- secdirectory
- srvrconf
- replist
- xattrschema

More Detailed Information — Creating and Maintaining Attribute Instances

Attaching Attribute Instances to Objects

You can attach attributes to objects when you create the objects with the **dcecp principal –attribute** operation or you can attach attributes to existing objects with the **dcecp modify –add** operation.

Modifying Attribute Instances

Use the **dcecp modify –change** operation to change the values of attribute instances. Whether an attribute is modifiable is determined by the application that uses the attribute.

If you use the **dcecp modify –change** command as shown in the previous paragraphs to change the value of a multivalued attribute, all instances of the multivalued attribute are deleted and replaced by the new values specified in the command.

You can add and remove individual values from a multivalued attribute. Use the **–add** option to add values.

Use the **remove** option described in the following section to delete specific values in a multivalued attribute.

If the **multi_name** attribute had the following values:

```
{multi_name value1 value2 value3}
```

The **dcecp modify** command changes the values to:

```
{multi_name value1}
```

Deleting Attribute Instances

Use the **dcecp modify** command with the **–remove** option to delete attribute instances attached to an object. To delete all instances of an attribute from an object, supply the attribute name to the **–remove** option.

To remove a single instance of a multivalued attribute, supply the attribute name and the attribute value.

If you delete the last instance of a multivalued attribute, **dcecp** also deletes the attribute from the object, because an attribute without a value cannot be attached to an object. Note you cannot combine deleting a mutlivalued attributes and values from multivalued attributes with the same command.

To delete more than one attribute from an object, you must use the **–types** option. This option tells **dcecp** that all the values supplied are the names of attribute types, not attribute values.

Using Attribute Sets

An attribute set is a collection of attribute UUIDs that identify the attribute instances that are members of the set. Attribute sets let you group related attributes instances on an object for easier access. For example, if you use the **dcecp show** operation to display an attribute set, the display expands the attribute set and includes all members of set in the display output. This attribute expansion works only for **dcecp** commands that display information. The commands to create an modify attribute instances work only on the specific attribute named in the command. Since the attributes that are set members exist independently of the attribute set, they can be manipulated directly like any other attribute.

Each attribute set is attached to an object and, although the system does not enforce it, each attribute that is a member of a set should also be attached to the same object. Attribute sets cannot be nested: a member of an attribute set cannot itself be an attribute set.

To create, modify, and delete members in an attribute set, follow the instructions to create, modify, and delete mutlivalued attributes. The attribute instances that are members of the set are identified by UUIDs.

Examples — Creating and Maintaining Attribute Instances

- To create the principal **delores** and at the same time attach the **MVSname** attribute with a value of **admin**, use the following **principal create** command:

  ```
  dcecp> principal create delores -attribute {MVSname admin}
  dcecp>
  ```

- To attach the **MVSname** attribute with a value of **admin** to the principal named **delores**, use the following **principal modify** command:

  ```
  dcecp> principal modify delores -add {MVSname admin}
  dcecp>
  ```

- To add instances of a multivalued extended attribute, include each value, separated by a space after the attribute name. For example, to attach the **multi_name** attribute with values of **value1**, **value2**, **value3**, and **value4** to the principal named **delores**, use the following command:

  ```
  dcecp> principal modify delores -add {multi_name value1 value2 value3 value4}
  dcecp>
  ```

- The following command changes the value assigned to the **MVSname** from **admin** to **cell_admin** for the principal named **delores**.

  ```
  dcecp> principal modify delores -change {MVSname cell_admin}
  dcecp>
  ```

- To change only a specific value, you must enter all the values. Assume that the **multi_name** attribute has the following four values — **value1**, **value2**, **value3**, and **value4**. To change **value4** to **value5**, you must enter the following command:

  ```
  dcecp> principal modify delores -change {multi_name {value1 value2 value3 value5}}
  dcecp>
  ```

- Assume that the **multi_name** attribute has values of **value1**, **value2**, **value3**, and **value5**. The following sample command adds **value6** to the **multi_name** attribute.

  ```
  dcecp> principal modify delores -add {multi_name value6}
  dcecp>
  ```

- The following command replaces all instances of the attribute named **multi_name** attached to the principal named **delores** with a single instance with a value of **value1**.

  ```
  dcecp> principal modify delores -change {multi_name value1}
  dcecp>
  ```

- The following command deletes all instances of the **MVSname** attribute from the principal named **delores**.

  ```
  dcecp> principal modify delores -remove MVSname
  dcecp>
  ```

- The following command deletes only the instance **value5** from the multivalued attribute named **multi-value**. All other values and the attribute itself remain intact.

  ```
  dcecp> principal modify delores -remove {multi-value value5}
  dcecp>
  ```

- The following sample command uses the **–types** option to delete the attributes named **MVSname** and **MVSinteger** from the principal **delores**.

  ```
  dcecp> principal modify delores -remove {MVSname MVSinteger} -types
  dcecp>
  ```

  Without the **–types** option **dcecp** will assume that **MVSinteger** is the value for the **MVSname** attribute and, because no such value exists, the command will fail.

Creating and Maintaining Attribute Types

Introduction — Creating and Maintaining Attribute Types

Use the **dcecp xattrschema** command to create and modify attribute types. When you use this command you must supply the attribute type's fully qualified name (for example, /**.:/sec/xattrschema**/*name*) as an argument.

More Detailed Information — Creating and Maintaining Attribute Types

Creating Attribute Types

Use the **dcecp xattrschema create** command to create attribute types. The syntax of this command is as follows:

xattrschema create *attr_name* {*attr_options*}

where:

attr_name Is the fully qualified name of the attribute type to create.

attr_option Is one or more of the options described below.

When you use this command you can supply the attribute options summarized in the following list. Note that the options described in this table can also be supplied without the hyphens in attribute lists.

Option	Meaning
–aclmgr *description*	A required list of the ACL Manager Types that support the objects to which this attribute type can be attached and the permissions supported by those managers. There is no default.
–annotation *string*	A PCS text string that annotates the attribute type. If the string contains spaces, enclose it in braces or quotes. The default is blank.
–applydefs	This option is not currently implemented.
–encoding *type*	The format of the attribute type instance value.
–intercell	This option is not currently implemented.
–multivalued {yes|no}	An indication of whether the attribute is multivalued (**yes**=multivalued, **no**=not multivalued). If an attribute is multivalued, multiple instances of the same attribute type can be attached to a single Registry object. For example, if Attribute A is encoded as multivalued, a single principal can have multiple instances of Attribute A. If it is not coded as multivalued, a single principal can have only one instance of Attribute A. The default is **no**.
–reserved {yes|no}	An indication of whether the attribute is reserved (**yes**=reserved, **no**=not reserved). Reserved attribute types cannot be deleted unless the reserved restriction is removed. The default is **no**.
–scope *name*	This option is not implemented in the current release.
–trigtype *type*	Identifies whether a trigger server is associated with the attribute type and, if a trigger server is associated, the type of trigger. Possible values are: **none** A trigger server is not associated with the attribute type. **query** A query trigger server is associated with the attribute type. **update** An update trigger server is associated with the attribute type. If the **–trigtype** option is set to **query** or **update**, you must supply the **–trigbind** option to specify the trigger server's binding. See “Defining Attribute Trigger Servers” on page 7-62 for more information on trigger servers.
–trigbind *binding*	If a trigger server is associated with the attribute type, this option specifies the trigger serving binding.
–unique {yes|no}	An indication of whether each instance of the attribute type must be unique within the cell (**yes**=unique, **no**=not unique). For example, assume that an instance of Attribute A is attached to 25 principals in the cell. If Attribute A is coded as unique, the value of the Attribute A for each of those 25 principals must be different. If it is not coded as unique, all 25 principals can be assigned the same value for Attribute A. The default is **no**.
–uuid *uuid*	An identifier for the attribute type that is used internally. Note that the name supplied as an argument to the **dcecp xattrschema create** command is used to access the attribute type. If you do not supply a UUID, the system generates one.

Modifying Attribute Types

Use the **dcecp modify** command with the **–change** option to modify attribute types. Only the **aclmgr**, **applydefs**, **intercell**, **trigbind**, **annotation**, and **reserved** schema type attributes can be modified.

The syntax of the **xattrschema modify** command is as follows:

xattrschema modify *attr_name* **-change** *new_option*

where:

attr_name Is the fully qualified name of the attribute type to change.

new_option Is the option that specifies the changes.

Renaming Attribute Types

Use the **dcecp xattrschema rename** command to change the name of an extended attribute. Enter the command in the following form:

xattrschema rename *old_name* **-to** *new_name*

where:

old_name Is the fully qualified extended attribute name to be changed.

new_name Is the new fully qualified extended attribute name.

Deleting Attribute Types

Use the **dcecp xattrschema delete** command to delete an extended attribute. Be aware that when you delete an attribute type you also delete all instances of that attribute type. For example, assume that an instance of the **MVSname** attribute is attached to a principal named **delores**. If you delete the **MVSname** attribute, you also delete the instance of that attribute attached to **delores**.

To delete attribute types enter the command in the following form:

xattrschema delete *attribute_name*

attribute_name is the fully qualified name of the attribute to be deleted.

Defining the ACL Managers for Attributes

When you define an extended attribute type, you must define the objects to which the attribute can be attached and the permissions to access the attribute. To this, you associate an attribute type with one or more ACL managers, and you supply the permission sets that control access to attribute instances of that type. The attribute can be attached only to the objects that are supported by the ACL Manager types named in its ACL Manager set. And, only the permissions named in the ACL Manager set are valid for accessing the attribute instance. (Note that these permissions are in addition to the permissions already established by the ACL manager for the object it controls.) For example, suppose an ACL manager set for an attribute type named **MVSname** lists only the ACL Manager type for principals. Then, instances of the attribute type named **MVSname** can be attached only to principals and not any other registry objects. The ACL Manager set for the **MVSname** attribute also contains the permissions that control access to the **MVSname** attribute.

Use the **dcecp xattrschema -aclmgr** option to specify an attribute's ACL manager set. This option has the form:

{*mgr_uuid queryset updateset testset deleteset* }

where:

mgr_uuid Is the UUID that identifies the ACL manager to be associated with the attribute type. You can supply either the UUID or one of the following shorthand names (which are converted internally to a UUID) to access the ACL manager types provided by the DCE:

policy	Accesses the ACL manager for the policy object
principal	Accesses the ACL manager for principals
group	Accesses the ACL manager for groups
organization	Accesses the ACL manager for organizations
secdirectory	Accesses the ACL manager for directories in the registry database
replist	Accesses the ACL manager for the replica list
xattrschema	Accesses the ACL manager for the registry schema
srvrconf	To access the ACL manager for the dced object.

queryset Is the permission set to query instances of the attribute.

updateset Is the permission set to modify instances of the attribute.

testset Is the permission set to test instances of the attribute.

deleteset Is the permission to delete instances of the attribute.

To enter a permission set with more than one permission, concatenate the permissions. For example, to enter the permissions **t**, **M**, and **d**, enter `tMd`.

Enclose each ACL manager type's information in braces and leave a space between each item (except, of course, between items in the concatenated permission sets).

Consider the following command to define an addition ACL Manager for the **MVS_name** attribute:

```
dcecp> xattrschema modify /.:/sec/xattrschema/MVS_name
                  -aclmgr {18dbdad2-23df-11cd-82d4-080009251352 r w t mD}
dcecp>
```

The command adds an ACL Manager identified by the UUID **18dbdad2-23df-11cd-82d4-080009251352** to the **MVS_name** attribute. The permissions sets for the ACL manager are as follows:

- **r** is the query permission set
- **w** is the update permission set
- **t** is the test permission set
- **mD** is the delete permission set

You cannot modify or delete an attribute type's ACL manager set. However, you can add additional manager types to it.

Defining the Attribute Type Encoding

You must define the format of values that can be supplied for an attribute type in the attribute type's encoding. An attribute can be assigned only those values that are in the format defined in the encoding. For example, the encoding can specify that instances of this attribute type contain values only in the form of UUIDs.

Each attribute type can have only one encoding and that encoding cannot be modified. In addition, a special encoding type lets you create attribute sets.

Use the **dcecp xattrschema -encoding** option to specify attribute encoding. This option has the form:

-encoding *type*

The *type* parameter is one of the encoding types described in the following list.

Encoding Type	Meaning
any	This type is not implemented in this release of DCE.
attrset	The attribute value must be a list of attribute type UUIDs, enclosed in braces. This encoding type defines an attribute set. Attribute sets permit easier attribute search and retrieval. For instance, a query on an attribute set returns all instances of attributes that are members of the set.
binding	The attribute value must consist of authentication, authorization, and binding information suitable for communicating with a DCE server. Use this encoding if you want to store a name or server binding as an object's attribute.
byte	The attribute value must be a string of bytes. The byte string is assumed to be a pickle or is otherwise a self-describing type. Note that this encoding type allows entry of binary data. See the **xattrschema** reference page for more information on entering binary data.
confidential	This type is not implemented in this release of DCE.
i18ndata	The attribute value must be an internationalized string of bytes with a tag identifying the OSF-registered codeset used to encode the data. Note that this encoding type allows entry of binary data. See the **xattrschema** reference page for more information on how to enter binary data.
integer	The attribute value must be a signed 32-bit integer.
printstring	The attribute instance value must be a character string printable by the DCE Portable Character Set (PCS).
stringarray	The attribute value must be an array of one of more printstrings. Note that the printstring can be a null.
uuid	The attribute value must be a DCE UUID.
void	The attribute has no value. It is only a marker that is either present or absent.

Examples — Creating and Maintaining Attribute Types

- The following sample command creates the extended attribute type named **employee_num** and assigns it an ACL Manager of **principal** and an encoding type of **integer**.

```
dcecp> xattrschema create /.:/sec/xattrschema/employee_num
                         -aclmgr principal -encoding integer
dcecp>
```

  Although the sample above uses options to supply information, you can use standard **dcecp** attribute lists.

  Note: Note that you can supply a list of names to create multiple schema entries with one operation. However, you should be aware that if the command argument contains more than one schema name, you cannot specify a UUID attribute and the attributes you specify are applied to all entries created.

- The following command modifies the **MVSname** attribute to change its annotation. Note that the fully qualified attribute type name must be supplied to the command.

```
dcecp> xattrschema modify /.:/sec/xattrschema/MVSname -change
                         {annotation {Use with version 2.3}}
dcecp>
```

- The following example shows the **xattrschema rename** command used to change an attribute name from **log_name** to **MVSname**.

```
dcecp> principal rename /.:/sec/xattrschema/log_name
                        -to MVSname
dcecp>
```

- To delete the extended attribute named **MVSname**, the command would be:

```
dcecp> xattrschema delete /.:/xattrschema/MVSname
dcecp>
```

Creating and Maintaining Filters

Introduction — Creating and Maintaining Filters

After starting the Audit daemon and creating the event class file, you can run the DCE control program to create, modify, or display the filters maintained by the Audit daemon. Use the **audfilter create**, **audfilter modify**, and **audfilter delete** commands to create, modify, and delete the filters. Use the **audfilter catalog** and **audfilter show** commands to display the existing filters.

More Detailed Information — Creating and Maintaining Filters

Creating Filters

Use the **audfilter create** command to create a filter. The **–attributes** option is required. The argument to the options is a filter guide or list of guides. Each filter guide is made up of three elements, an event class name (or list of names), an audit condition (or list of conditions), and an audit action (or list of actions).

The event class name corresponds to the name of the event class file for which your are creating a filter.

The audit condition is the condition required for the event to be audited. Valid conditions are **success**, **denial**, **failure**, **pending**, and **all**.

The audit action is the action to take if the event being generated matches the audit condition specified. Valid actions are **log**, **alarm**, and **all**.

Modifying Filters

You can modify an existing audit filter by adding or deleting one or more of the filter's guides.

dcecp does not use commas. Multiple guides and multiple filters are specified in the standard **dcecp** list format — `{x y}` for single arguments or `{{x y} {a b}}` for multiple arguments.

In order to run the **audfilter modify** command, you must have write (**w**) permission to the Audit daemon's ACL.

Deleting Filters

You can delete one or more of the audit filters for a DCE client by using the **audfilter delete** command.

You can specify more than one filter to be operated on in the **audfilter delete** command. As with the example of modifying filters above, when deleting multiple filter, you must use the standard **dcecp** syntax.

In order to execute this command, you must have write (**w**) permission to the Audit daemon's ACL.

Default Filters

During the configuration of the host, the following **audfilter create** commands (using the DCE control program) are run to create filters for the Security daemon, DTS daemon, and the audit daemon:

```
audfilter create world -at {dce_sec_modify success log}
audfilter create world -at {dce_sec_modify {failure denial} all}
audfilter create world -at {dce_sec_server success log}
audfilter create world -at {dce_sec_server {failure denial} all}
audfilter create world -at {dce_sec_authent {failure denial} all}
audfilter create world -at {dce_sec_query denial all}
audfilter create world -at {dce_dts_mgt_modify success log}
audfilter create world -at {dce_dts_mgt_modify {failure denial} all}
audfilter create world -at {dce_dts_mgt_query {failure denial} all}
audfilter create world -at {dce_audit_admin_modify success log}
audfilter create world -at {dce_audit_admin_modify {failure denial} all}
audfilter create world -at {dce_audit_filter_modify success log}
audfilter create world -at {dce_audit_filter_modify {failure denial} all}
audfilter create world -at {dce_audit_admin_query {failure denial} all}
audfilter create world -at {dce_audit_filter_query {failure denial} all}
```

Enabling Audit Filters

If you want to enable the Audit filters, you must first set the **DCEAUDITFILTERON** environment variable. You must set this variable before starting the server (that is, the Audit client).

Removing the Update Binding File

If a server (Audit client) is running with filters enabled (that is, **DCEAUDITFILTERON** was set), **libaudit** (which is linked to the server) obtains the server's binding information and stores it in:

/opt/dcelocal/var/audit/client/*pid-of-server***/update_binding_file**

where *pid-of-server* is the process ID of the server.

If the server ends abnormally, this file must be removed manually. If this is not removed, you will receive an error message the next time you restart the server with **DCEAUDITFILTERON**. The message:

```
unable to inform process
/opt/dcelocal/var/audit/client/pid-of-server/update_binding_file
about esl update.
```

indicates that the Audit daemon is unable to inform the Audit client of filter updates. You can also check for stale update binding files by checking what servers are running (for example, using **ps –e**) and comparing their process IDs with the path names of the update binding files. Because the path name of these files contain a *pid-of-server* component, you can determine what files correspond to non-existent servers.

Both the binding information file and the directory containing it (*pid-of-server*) must be removed.

Buffering of the Audit Trail

The operating system buffers the audit trail data while it is written before writing it to disk. For this reason, the growth of the audit trail file will not become apparent until the data is flushed to disk.

Examples — Creating and Maintaining Filters

- The following is an example **audfilter create** command for creating a filter.

  ```
  dcecp> audfilter create {group trust} -attribute {ec_local_bank_audit denial log}
  ```

The example command specifies that a filter type **group** be created for the DCE group named **trust** in the local cell.

- The following is an example **dcecp** command for modifying an existing filter:

  ```
  dcecp> audfilter modify world -add {Monetary_Transfers denial log}
  ```

  The example command adds a guide with an event class of **Monetary_Transfers**, an audit condition of **denial**, and an audit action of **log** to the existing filter type **world**. Note that the filter type **world** does not take a key.

- The following is an example **audfilter delete** command:

  ```
  dcecp> audfilter delete  {foreign_principal /.../foreign_cell_name/jedwards}
  ```

  The example command deletes the audit filter for the DCE principal **jedwards** in the foreign cell **/.../foreign_cell_name**.

Creating Slave Replicas

After the master replica database has been created and started and its database has been populated, configure the secondary sites to create the slave replicas and start them. The configuration process ensures that the sites are running **dced** and the appropriate CDS servers. See *DSS for AIX Up and Running!* for detailed information on the configuration process.

Creating the Primary Registry Database

Introduction — Creating the Primary Registry Database

When you initially configure the cell Security server, the process invokes the **/opt/dcelocal/bin/sec_create_db** command to create the master replica. When **sec_create_db** creates a new master replica, it initializes its database with names and accounts. Note that you must be **root** to run **sec_create_db**.

The **sec_create_db** command also creates a registry configuration file, which is named **/opt/dcelocal/etc/security/pe_site**, that contains the cell name and network address of the master replica. This file supplies the binding address of the **secd** server to clients running on that machine if the Cell Directory Service is unavailable.

If you ever need to create a new primary registry database, you can invoke **sec_create_db** directly. While it is highly unusual to recreate a primary database, you may need to recreate a secondary database if the secondary is destroyed. Note that you must be **root** to run **sec_create_db**. It is also highly unusual to recreate a primary database, but you may need to recreate a secondary database if the secondary is destroyed. The following describes **sec_create_db** command usage.

More Detailed Information — Creating the Primary Registry Database

The sec_create_db Command Format

The **sec_create_db** command has the following format:

```
sec_create_db –my[name] my_server_name
              [–k[eyseed] keyseed]
              [–cr[eator] creator_name]
              [–cu[nix_id] creator_unix_id]
              [–u[uid] cell_unix_id]
              [–pe[rson_low_unix_id] unix_id]
              [–g[roup_low_unix-id] unix_id]
              [–o[rg_low_unix-id] unix_id]
              [–ma[x_unix_id] unix_id]
              [–pa[ssword] default_password]
              [–v[erbose]]
              [–slave|–master]
              [–h[elp]]
```

where:

–my[name] *my_server_name*
Represents a name that you assign to the Security server (**secd**) on this machine. It is used by the name service to locate this cell's security server (required).

–k[eyseed] *keyseed* Represents a character string that you enter to seed the random key generator in order to create the master key for the database that you are creating. It should be a string that cannot be easily guessed. The master key is used to encrypt all account passwords. Each instance of a replica (master or slave) has its own master key. You can change the master key by using the **dcecp registry modify** or **sec_admin master_key** command.

–cr[eator] *creator_name*
Represents the name of the registry creator. The registry creator is the initial privileged user of the registry database. Note that you can give equivalent privileges to another user at any time by using the **dcecp acl modify** or **acl_edit modify** command to change the registry database ACL. When the registry is created, default ACL entries for registry objects are also created. These entries give the most privileged permissions to the principal that is named in the **–cr** option. If the principal that is named as the registry creator is not one of the reserved names, **sec_create_db** adds the principal and an account for that principal. If you do not enter this option, the initial privileged user of the registry database is **root**.

–cu[nix_id] *creator_unix_id*
Represents a UNIX number that you specify to be assigned to the registry creator. If you do not enter this option, the registry creator's UNIX number is assigned dynamically.

–u[uid] *cell_uuid* Represents the cell's UUID. If you do not enter this UUID, it is assigned dynamically.

–pe[rson_low_unix_id] *unix_id*
Indicates the starting point for UNIX IDs that are automatically generated by when a principal is added by using the **dcecp registry modify** or **rgy_edit properties** command. Note that you can explicitly assign a lower UNIX ID than this number. This lower limit applies only to automatically-generated UNIX IDs.

–g[roup_low_unix_id] *unix_id*
Indicates the starting point for UNIX IDs that are automatically generated when a group is added using the **dcecp registry modify** or **rgy_edit properties** command. Note that you can explicitly enter a lower UNIX ID than this number. This lower limit applies only to automatically-generated UNIX IDs.

–o[rg_low_unix_id] *unix_id*
Indicates the starting point for UNIX IDs that are automatically generated by the Security Service when an organization is added by using the **rgy_edit** command. Note that you can explicitly enter a lower UNIX ID than this number. This lower limit applies only to automatically generated UNIX IDs.

–ma[x_unix_id] *unix_id*
Represents the highest number that can be assigned as a UNIX ID when a principal, group, or organization is added. No UNIX IDs higher than this number are assigned automatically and you cannot specifically enter numbers higher than this number. The maximum UNIX ID stays in place until you change it with the **dcecp registry modify** or **rgy_edit properties** command.

–pa[ssword] *default_password*
Represents the default password that is assigned to the accounts created by **sec_create_db**. If you do not specify a default password, **-dce-** is used. Note that the **hosts**/*local_host_name*/*principal_name* **none none**, **krbtgt**/*cell_name* **none none**, and **nobody none none** accounts are not assigned the default password, but instead are assigned a randomly-generated password.

–v[erbose] Runs in verbose mode and generates a verbose transcript of all activity.

–slave Specifies that a slave replica's database should be created. Only the **–myname**, **–keyseed**, and **–verbose** options can be used with the **–slave** option.

–master Specifies that the master replica's database should be created. All other **sec_create_db** options can be used with the **–master** option.

–h[elp] Displays usage syntax.

The Results of sec_create_db

The primary registry database that is created by **sec_create_db** contains the principals, groups, and organizations listed in Table 7-4.

Table 7-4. Initial Persons, Groups, and Organizations

Principal	Group	Organization
bin	bin	none
daemon	daemon	—
dce-ptgt	kmem	—
dce-rgy	mail	—
krbtgt/*local_cell_name*	nogroup	—
hosts/*local_host*/self	none	—
mail	system	—
nobody	tcb	—
root	tty	—
sys	uucp	—
tcb	—	—
uucp	—	—
who	—	—

The accounts that are created by **sec_create_db** are as follows:

- **bin bin none**
- **daemon daemon none**
- **dce-ptgt none none**
- **dce-rgy none none**
- **hosts**/*local_host*/**self none none**
- **krbgt**/*cell_name* **none none**
- **nobody nogroup none**
- **root system none**
- **uucp uucp none**

Some of the objects that were initially created by **sec_create_db** are reserved and cannot be deleted.

- The reserved principals are:
  - **dce-ptgt**
  - **krbtgt**/*cell_name*
  - **dce-rgy**
- The reserved group is **none**.
- The reserved organization is **none**.
- The reserved accounts are:
  - **dce-ptgt none none**
  - **krbtgt**/*cell_name* **none none**
  - **dce-rgy none none**

When you run the **sec_create_db** command to create the primary registry database, you can name the principal who has the most privileged access to the registry. This person is known as the registry creator. If the registry creator you name is not one of the default principals, **sec_create_db** adds the account *rgy_creator* **none none** (where *rgy_creator* is the principal you named as the registry creator). If you do not name a registry creator, **sec_create_db** assigns the most privileged registry access to the **root system none** account.

With one exception, all of the accounts created by the **sec_create_db** command are assigned randomly-generated passwords and are marked as invalid. Before these principals can log into these accounts, you must change the account passwords and mark the accounts as valid. You can do this by using the **dcecp account modify** or **rgy_edit change** command. You can use the **dcecp account modify** command to change all attributes for a principal's account in the registry, including the principal's password. Also, both commands have options to randomly generate new passwords. However, the exception is that the account created for the registry creator is valid and is assigned the DCE default password (**-dce-**). Change the default password to ensure the security of the registry creator account.

In addition to the group memberships implied by the accounts that are created by **sec_create_db**, the principals are also made members of the groups listed below:

The principal...	Is a member of the group...
who	**bin**
root	**system** **kmem** **tty**
sys	**kmem**
mail	**mail**
tcb	**tcb**

Example — Creating the Primary Registry Database

The following example shows the **sec_create_db** command that is run to create the primary database and the information that **sec_create_db** displays as it runs. Note that, because the **–k** option is not entered, **sec_create_db** prompts you for the master key seed string. This string is not displayed as it is entered.

```
$ /work/krb/sec_create_db -v -myname /.../dresden.com/subsys/dce/sec/master -master

Enter keyseed for initial database master key: <enter up to
      1024 characters>
SECD Checkpoint on Tue Sep 27 11:44:12 1994
.... saving rgy
.... saving acct
.... saving person
.... saving group
.... saving org
.... saving replicas
.... saving acl
End SECD Checkpoint on Tue Sep 27 11:44:13 1994
SECD Checkpoint on Tue Sep 27 11:44:15 1994
.... saving rgy
.... saving acct
.... saving person
.... saving group
.... saving org
.... saving acl
End SECD Checkpoint on Tue Sep 27 11:44:17 1994
```

Creating Trust Relationships

Introduction — Creating Trust Relationships

Use the **registry connect** and **account modify** commands to create trust relationships.

More Detailed Information — Creating Trust Relationships

Command Options for the registry connect Command

When you use the **registry connect** command, you must supply the fully qualified name of the foreign cell with which you will establish a trust relationship. This name is stripped of the full path name, prefixed with **krbtgt**, and used as the primary name of the account's principal. For example, if you enter a cell name of /**.../dresden.com**, the principal name is **krbtgt**/**dresden.com**. The unchanged cell name is stored as the principal's full name.

Note that **registry connect** uses your local cell name for the primary name of the local cell's account principal. This name is stripped of the full path name and prefixed with **krbtgt**, as is the foreign cell name.

The following list describes additional information that you can supply to the **registry connect** command.

Option	Meaning
–mypwd	The **registry connect** does not prompt you for a password for the accounts that you are creating; it generates this password randomly. However, you must supply your password with the **mypw** option to validate your identity.
–facct and **–facctpw**	The system administrator in the foreign cell must provide the name and password of an account in the foreign cell. The foreign account must have the permissions that are required to create principals and accounts. You need the account to create principals and accounts. You need the account to access the foreign registry in order to create the account that represents your cell in the foreign account's registry. The lifetime and creation quota of this account should be limited to only that necessary to complete this task.
–group and **–fgroup**	The group name to be associated with the account in the local cell (**–group**) and the foreign cell (**–fgroup**). These groups have no meaning for the accounts and are not associated with any users in the foreign or local cell. You must enter them because it is a requirement of the registry that all accounts be associated with groups. If the group does not exist, it is created.
–org and **–forg**	The organization name to be associated with the account in the local cell (**–org**) and the foreign cell (**–forg**). These organizations have no meaning for the accounts and are not associated with any users in the foreign or local cell. You must enter them because it is a requirement of the registry that all accounts be associated with organizations. If the organization does not exist, it is created.
–expdate	The time and date that both the local and foreign cell's account expires, and the trust relationship is ended, prohibiting any further authenticated communications between principals in the two cells. To renew the account, change the date in this field. The default is **none**.

The Accounts Created by the registry connect Command

The accounts and principals that are created by the **registry connect** command are given default attribute values listed below. These attributes apply to all foreign principals when they access objects in your cell. Likewise, the attributes of the account created for your cell in the foreign cell apply to all principals in your cell when they access objects in the foreign cell.

Information	Meaning
Account Principal Name	The local cell name for the local cell's account, or foreign cell name for the foreign cell's account, stripped of its full path name and prefixed with **krbtgt**.
fullname	The cell's path name.
quota	Set to **none**. This quota applies to all principals who use the cross-cell authentication accounts to access objects in foreign cells. This means that, for example, if you change the object creation quota to 10, the total number of objects that can be created in your cell's registry by all foreign users who use the account to access your cell cannot exceed 10. It is not 10 per foreign principal. The object creation quota that is set for your cell's account in the foreign cell places the same restriction on the number of objects that your cell's principals can create in the foreign cell's registry.

description home shell
Set to blank.

server Set to **yes**; that is, the account is a server that can engage in authenticated communications.

client Set to **no**.

pwdvalid Set to **yes** (valid).

acctvalid Set to **no** (not valid).

postdatedtkt Set to **yes**; that is, the account can be issued tickets with a start time in the future.

forwardabletkt Set to **yes**; that is, the account can be issued a new ticket-granting ticket with a network address that is different than the present ticket-granting ticket.

renewabletkt Set to **yes**; that is, the account's tickets can be renewed.

proxiabletkt Set to **yes**; that is, the account can be issued tickets with a different network address than the present tickets.

dupkey Set to **yes**; that is, the account's ticket can have duplicate keys.

goodsince Set to the date that the account was created.

maxtktlife Set to the registry policy.

maxtktrenew Set to the registry policy.

Steps — Creating Trust Relationships

To create trust relationships, follow these steps:

1. Run the **registry connect** command to create cross-cell authentication accounts (an account in your cell's registry and another account in the foreign cell's registry with the same name as the cell).
2. Optionally, use the **account modify** command to fine tune the account attributes, which were assigned by default when the account was created. For example, if the account's expiration date (**expdate** attribute) defaults to **none**, you may want to enter a date to ensure the account will be actively renewed after a period of time.
3. Ensure that the system administrator in the foreign cell changes the **acctvalid** flag of the account that represents your cell to **yes** in order to indicate that the account is valid. If one or both accounts are invalid, no cross-cell communication can take place.

Example — Creating Trust Relationships

The following sample **registry connect** command is used to create an account for the foreign cell identified by /**.../dresden.com**. The local account is associated with the local group named **cell_group_local**, the foreign group named **cell_group_dres**, the local organization named **cell_org_local**, and the foreign organization named **cell_org_dres**. The expiration date for the accounts is allowed to default to **none**.

```
dcecp> registry connect /.../dresden.com
                          -facct cell_log -facctpw music
                          -group cell_group_local -fgroup cell_group_dres
                          -org cell_org_local -forg cell_org_dres
                          -mypwd cell_admin
dcecp>
```

Defining Attribute Trigger Servers

Introduction — Defining Attribute Trigger Servers

Some attribute types require the support of an outside server either to verify input attribute values or to supply output attribute values when those values are stored in an external database. Such a server could, for example, connect a legacy registry system to the DCE Registry. The attribute trigger facility provides for automatic calls to outside DCE servers, known as attribute triggers.

More Detailed Information — Defining Attribute Trigger Servers

Trigger servers are invoked automatically when an attribute associated with a trigger server is queried or updated. Note that access to information maintained by a trigger server is controlled entirely by that server.

Note: Update trigger servers are not supported in this release.

To associate an attribute type with a trigger server, use the **–trigtype** and **–trigbind dcecp xattrschema** options.

The **–trigtype** options defines whether the attribute type is associated with a trigger server and if it is, which kind of server. This option has the form:

-trigtype [none | query | update]

where:

none Indicates the attribute is not associated with a trigger server. (This is the default.)

query Indicates that the attribute is associated with a query trigger. Query trigger servers can perform only queries.

update Indicates the attribute is associated with an update trigger. Update trigger servers can perform queries and updates.

Note: Update trigger servers are not supported in this release.

After it is set, the **–trigtype** option cannot be modified.

The –trigbind Option

The **–trigbind** option defines authentication information for the trigger server and the trigger binding itself.

The **–trigbind** option has the following format.

-trigbind {{*auth_info*} {*binding_info*}**}**

The following sections describe how to specify the authentication type and the binding.

Specifying the Authentication Type

The *auth_info* parameter has the following syntax:

{*auth_serv_type name prot_level authentication_service authorization_service*}

where:

auth_serv_type

Specifies the authentication type, which can be:

none No authentication is performed.

dce Standard DCE authentication is performed.

If you are using no authentication, no other information except the binding itself is required. If you are using the standard DCE authentication type, you must specify all the remaining parameters.

name Specifies the principal name of the trigger server.

prot_level Specifies the protection level that determines the degree to which authenticated communications between the client and the server are protected by the authentication service. The possible protection levels are:

default Uses the default protection level of **pkt**.

none Performs no authentication — tickets are not exchanged, session keys are not established, client EPACs or names are not certified, and transmissions are in the clear. Note that although uncertified EPACs should not be trusted, they may be useful for debugging, tracing, and measurement purposes.

connect Authenticates only when the client establishes a relationship with the server.

call Authenticates only at the beginning of each remote procedure call when the server receives the request.

This level does not apply to remote procedure calls made over a connection-based protocol sequence (that is, **ncacn_ip_tcp**). If this level is specified and the binding handle uses a connection-based protocol sequence, the routine uses the **pkt** protection level instead.

pkt Ensures that all data received is from the expected client.

pktinteg Ensures and verifies that none of the data transferred between client and server has been modified. This is the highest protection level that is guaranteed to be present in the RPC runtime.

pktprivacy Authenticates as specified by all of the previous levels and also encrypts each RPC argument value. This is the highest protection level, but is not guaranteed to be present in the RPC runtime.

authentication_service

Specifies the authentication service. The exact level of protection provided by the authentication service is specified by the protection level. The supported authentication services are as follows:

default DCE shared-secret key.

none No authentication — no tickets are exchanged, no session keys established, client EPACs or names are not transmitted, and transmissions are in the clear. Specify **none** to turn authentication off for remote procedure calls made using this binding.

secret DCE shared-secret key authentication.

authorization_service

Specifies the authorization service. The validity and trustworthiness of authorization data, like any application data, is dependent on the authentication service and protection level specified. The supported authorization services are as follows:

none Server performs no authorization. This is valid only if the authorization service is set to **none**, specifying that no authentication is being performed.

name Server performs authorization based on the client principal name. This value cannot be used if the authorization service is **none**.

dce Server performs authorization using the client's DCE EPAC sent to the server with each remote procedure call made with this binding. Generally, access is checked against DCE Access Control Lists (ACLs).

Specifying the Binding Information

The *binding_info* parameter specifies the binding, which can be a string binding, a server entry name, or a list containing one or more string bindings or server entry names.

Examples — Defining Attribute Trigger Servers

- The following example shows a server entry name binding:

  ```
  ./.:/hosts/host_name/dce_entity_name
  ```

- The following example shows a string binding in standard syntax:

  ```
  ncadg_udp_ip:130.105.96.3[1234]
  ```

  The following example shows a string binding in TCL syntax:

  ```
  ncarn_ip_tcp 10-29.58.00 2001
  ```

- The following sample shows the value for a **–trigbind** option. In the sample, the binding has the principal name **MVS_server**, is authenticated with packet-privacy protection level, and uses an authorization service of DCE shared secret key and an authorization service of DCE. The binding is supplied as a server entry name.

  ```
  -trigbind {{dce MVS_server pktprivacy secret dce}
             {/.:/hosts/host_name/dce_entity_name}}
  ```

Defining Event Classes

Introduction — Defining Event Classes

Individual audit events can be grouped together to form event classes. The event class provides an efficient mechanism by which sets of events can be logically grouped and selected using a single value.

DCE Audit event classes are configurable. You can add or remove events of an existing event class or define new event classes.

The ability to define local event classes is useful in simplifying the management of audit services in multiple DCE applications. Administrators can design their own audit event classes reflecting their security requirements and trail storage resource constraints.

Temporary event classes can also be created to track down security violations.

Steps — Defining Event Classes

To define an event class, follow these steps:

1. Obtain an event class number for the event class from your cell administrator. A range of event class numbers should have been allocated to your organization by OSF. If not, contact OSF.
2. Create an event class file in the /**opt**/**dcelocal**/**etc**/**audit**/**ec** directory. Edit the file as follows:
   - Declare the event class number (ECN) by adding a line which has the following format:

     ECN=*_event_class_number*

- Optionally, you can add a Server Event Prefix (SEP) line in the file. The SEP line contains the event number prefixes of each server. The event number prefix is the lowest event number in each server. The SEP line has the following format:

  SEP=*_event_number1 event_number2 event_number3 ...*

  You can put the SEP line anywhere on the file. The SEP line speeds up the scanning of audit clients by skipping irrelevant event class files.
- From the application, obtain the event numbers for the code points that will be included in the event class.
- Add the event numbers corresponding to the events that you want to include in the event class, one number per line.

In the event class file, empty lines are ignored and comments are designated by the number sign (#) before the comment text.

Example — Defining Event Classes

The following is an example of an event class file named **ec_local_cell_critical_events**.

```
ECN = 0xC0000005
# Server Event Number Prefixes
# 0x000001 Security Service Events
# 0x000002 Time Service Events
# 0x000003 Audit Service Events
SEP = 0x00000100 0x00000200 0x00000300
# Security Service Critical Events
# evt_osf_dce_rs_properties_set_info (sets registry properties)
0x0000011f
# evt_osf_dce_rs_policy_set_info (sets registry policy)
0x00000121
# evt_osf_dce_rs_rep_admin_stop (stops the registery service)
0x00000127
# evet_osf_dce_rs_rep_admin_mkey (changes master key)
0x00000129
# Time Service Critical Events
# evt_osf_dce_dts_create (creates a server or a clerk)
0x00000201
# evt_osf_dce_dts_delete (deletes a server or a clerk)
0x00000202
# evt_osf_dce_dts_enable (enables the time service)
0x00000203
# evt_osf_dce_dts_disable (disables the time service)
0x00000204
# Audit Service Critical Events
# evt_osf_dce_aud_enable (enables audit-record logging service)
0x00000301
# evt_osf_dce_aud_disable (disables audit-record logging service)
0x00000302
# evt_osf_dce_aud_stop (terminates the execution of the audit daemon)
0x00000303
```

Displaying Account Information

Introduction — Displaying Account Information

Use the **dcecp account catalog** and **account show** commands to display information about accounts. When you use the **account show** command, you must supply the name of the account's principal to specify the account to display. You can supply multiple principal names by enclosing them in braces and separating them with spaces.

Steps — Displaying Account Information

- To display all accounts in the registry database in alphabetic order with names prefixed by the cell name, enter:

  account catalog

- To display all accounts in the registry database in alphabetic order with names *not* prefixed by the cell name, enter:

  account catalog -simplename

- To display all attributes for a named principal's account, enter:

  account show *principal_name*

- To display all policies for a named principal's account, enter:

  account show *acct_name*-**policy**

- To display all attributes and all policies for a named principal's account, enter:

  account show *acct_name*-**all**

Examples — Displaying Account Information

- The following example shows the **account catalog** used without the **–simplename** option.

```
dcecp> account catalog
/.../dresden.com/bach
/.../dresden.com/bin
/.../dresden.com/brahms
/.../dresden.com/britten
/.../dresden.com/cell_admin
/.../dresden.com/daemon
/.../dresden.com/dce-ptgt
/.../dresden.com/dce-rgy
/.../dresden.com/mahler
/.../dresden.com/nobody
/.../dresden.com/root
/.../dresden.com/uucp
/.../dresden.com/hosts/pmin17/cds-server
/.../dresden.com/hosts/pmin17/gda
/.../dresden.com/hosts/pmin17/self
/.../dresden.com/krbtgt/dresden.com
dcecp>
```

- The following example shows the **account show** command used to display the attributes and associated with the account for **mahler**.

```
dcecp> account show mahler
{acctvalid yes}
{client yes}
{created /.../dresden.com/cell_admin 1994-06-15-18:31:08.000+00:00I-----}
{description {}}
{dupkey no}
{expdate 1995-06-16-00:00:00.000+00:00I-----}
{forwardabletkt yes}
{goodsince 1994-06-15-18:31:05.000+00:00I-----}
{group users}
{home /}
{lastchange /.../dresden.com/cell_admin 1994-06-16-12:21:07.000+00:00I-----}
{organization users}
{postdatedtkt no}
{proxiabletkt no}
{pwdvalid yes}
{renewabletkt yes}
{server yes}
{shell {}}
{stdtgtauth yes}
dcecp>
```

  Note that if the policy defined for account is not actually in effect because it is overridden by the registry policy, policy is followed by the **effective** tag and the actual value in effect.

Displaying ACL Information

Introduction — Displaying ACL Information

Use the **dcecp acl show** commands to display a ACL entires for a named object. When you use this command you must specify the name of the object for which to display ACL entries. You can supply multiple names by enclosing them in braces and separating them with spaces.

If this command cannot determine the name of the object, it displays the object's UUID.

Steps — Displaying ACL Information

- To display the ACL entries for a specified object, enter:

  acl show *object_name*

- To display the ACL's default cell, enter:

  acl show *object_name* **-cell**

- To display the ACL managers supported by an object, enter:

  acl show *object_name* **-managers**

Example — Displaying ACL Information

The following example displays ACL entries for the object named **hosts**.

```
dcecp> acl show /.:/hosts
{unauthenticated r--t---}
{user cell_admin rwdtcia}
{user hosts/absolut/cds-server1 rwdtcia}
{user root rwdtcia}
{group subsys/dce/cds-admin rwdtcia}
{group subsys/dce/cds-server rwdtcia}
{any_other r--t---}
```

Displaying Group and Organization Information

Introduction — Displaying Group and Organization Information

Use the **dcecp group catalog**, **group show**, and **group list** commands to display information about groups and the **dcecp organization catalog**, **organization show**, and **organization list** commands to display information about organizations. When you use the **group list**, **group show**, **organization list**, and **organization show** commands, you must supply the name of the group or organization to display. You can supply multiple names by enclosing them in braces and separating them with spaces.

Steps — Displaying Group and Organization Information

- To display all groups or organizations in the registry database in alphabetic order with names prefixed by the cell name, enter:

  group catalog

  OR

  organization catalog

- To display all groups or organizations in the registry database in alphabetic order with names *not* prefixed by the cell name, enter:

  group catalog -simplename.

  OR

  organization catalog -simplename

- To display all members of a specified group or organization in alphabetical order with names prefixed by the cell name, enter:

  group list *group_name*

  OR

  organization list *group_name*

- To display all members of a specified group or organization in alphabetical order with names *not* prefixed by the cell name, enter:

  group list *group_name* **-simplename**

  OR

  organization list *group_name* **-simplename**

- To display all attributes for a group or organization, enter:

  group show *group_name*

  OR

  organization show *group_name*

- To display all extended attribute instances attached to a group or organization, enter:

  group show *group_name* **-xattrs**

OR

organization show *group_name* **-xattrs**

- To display all regular attributes and all extended attributes for a group or organization, enter:

  group show *group_name* **-all**

  OR

  organization show *group_name* **-all**

Examples — Displaying Group and Organization Information

- The following example shows the **group catalog** used without the **–simplename** option.

```
dcecp> group cat
/.../dresden.com/nogroup
/.../dresden.com/system
/.../dresden.com/daemon
/.../dresden.com/uucp
/.../dresden.com/bin
/.../dresden.com/kmem
/.../dresden.com/mail
/.../dresden.com/tty
/.../dresden.com/none
/.../dresden.com/tcb
/.../dresden.com/acct-admin
/.../dresden.com/subsys/dce/sec-admin
/.../dresden.com/subsys/dce/cds-admin
/.../dresden.com/subsys/dce/dts-admin
/.../dresden.com/subsys/dce/cds-server
/.../dresden.com/subsys/dce/dts-servers
/.../dresden.com/users
dcecp>
```

- The following example the attributes of the group named **users_temporary**:

```
dcecp> group show users_temporary
{alias no}
{gid 5211}
{uuid 0000145b-9362-21cd-a601-0000c08adf56}
{inprojlist no}
{fullname {temporary users}}
dcecp>
```

  Note: In the preceding example the line specifies `{alias no}`. This line indicates that the name **users_temporary** is the primary name, not an alias name. For an alias, this line would specify `{alias yes}`.

- The following **group list** command displays the members of the group **symphonists**.

```
dcecp> group list  symphonists
/.../dresden.com/bach
/.../dresden.com/britten
/.../dresden.com/mahler
```

Displaying keytab Information

Introduction — Displaying keytab Information

Use the **dcecp keytab catalog**, **keytab list**, and **keytab show** commands to display information about accounts. When you use the **keytab catalog** command, you must supply the name of the host for which to display keytab files. When you use the **keytab list** or **keytab show** command, you must supply the name of the **dced** object for which to display keytab information. You can supply multiple names to either command by enclosing them in braces and separating them with spaces.

Steps — Displaying keytab Information

- To display the names of all keytab files on a specified host with names prefixed by the cell name, enter:

  keytab catalog *host_name*

  If you do no supply a *host_name*, the display lists keytab files on the current host.

- To display the names of all keytab files on a specified host with names *not* prefixed by the cell name, enter:

  keytab catalog *host_name* **-simplename**

- To display a list of all principals for which there are entries in a specified keytab file, enter:

  keytab list *file_name*

- To display all principals for which there are entries in a named keytab file, enter:

  keytab show *file_name*

  The information displayed includes only the principal name.

- To display the local names of a specified key file, enter:

  keytab show *dced_object_name* **-entry**

- To display all entries in a key file, including the keys, enter:

  keytab show *dced_object_name* **-members**

Restrictions — Displaying keytab Information

All key types must be stored as **des**. If a key type is specified as **plain**, it is automatically converted to **des** before it is stored. For any country that does not allow or does not get **des**, the **–noprivacy** flag can be used instead.

Example — Displaying keytab Information

The following example shows the entries in the keytab file named **svr_3**:

```
dcecp> keytab show /.:/hosts/music/config/keytab/svr_3 -members
{brahms des 1}
{britten plain 3}
{mahler des 2}
dcecp>
```

Displaying Principal Information

Introduction — Displaying Principal Information

Use the **dcecp principal catalog** and **principal show** commands to display information about principals. When you use the **principal show** command, you must supply the name of the principal to display. You can supply multiple principal names by enclosing them in braces and separating them with spaces.

Steps — Displaying Principal Information

- To display all principals in the registry database in alphabetic order with names prefixed by the cell name, enter:

  principal catalog

- To display all principals in the registry database in alphabetic order with names *not* prefixed by the cell name, enter:

  principal catalog -simplename

- To display all attributes for a named principal, enter:

  principal show *principal_name*

- To display all extended attribute instances attached to a principal, enter:

  principal show *principal_name* **-xattrs**

- To display all regular attributes and all extended attributes for a principal, enter:

  principal show *principal_name* **-all**

Examples — Displaying Principal Information

- The following example shows the **principal catalog** used with the **–simplename** option.

```
dcecp> principal catalog -simplename
bach
bin
brahms
britten
cell_admin
daemon
dce-ptgt
dce-rgy
mahler
nobody
root
uucp
cds-server
dcecp>
```

- The following example shows the **principal show** command used to display information about the principal **mahler**.

```
dcecp> principal show /.:/mahler
{fullname {Gustav Mahler}}
{uid 30014}
{uuid 0000753e-f51f-2e0e-b000-0000c08adf56}
{alias no}
{quota unlimited}
{groups {{symphonists composers}}
dcecp>
```

  All the information listed by the **principal show** command is information created when principal was added to the registry, except the line for groups. This line lists the groups in which the principal is a member.

Displaying xattrschema Information

Introduction — Displaying xattrschema Information

Use the **dcecp xattrschema catalog** and **xattrschema show** commands to display information about the extended attribute types. To see instances of an extended attribute attached to a principal, use the **–xattr** option with the **principal**, **group**, or **organization show** commands.

The **xattrschema catalog** command displays the names of the extended attribute instances defined in a named schema. When you use this command you must specify the name of the schema for which to display extended attributes. For the registry data base, this name is /**.:/sec/xattrschema**. Your site must supply you with the name of schemas or the registry. The **xattrschema show** command displays the attributes of named schemas in either the registry schema or a schema in use at your site. When you use this command you must specify the name of the extended attribute type for which to display information. You can supply multiple names by enclosing them in braces and separating them with spaces.

Steps — Displaying xattrschema Information

- To display the names of all attribute types in the registry database in alphabetic order with names prefixed by the cell name, enter:

  xattrschema catalog /.:/sec/xattrschema

- To display all attribute types in the registry database in alphabetic order *not* prefixed by the cell name, enter:

  xattrschema catalog /.:/sec/xattrschema

- To display attributes in a schema other than the registry, replace /**.:/sec/xattrschema** with the fully specified name of the other schema.

- To display the attributes of a named extended attribute type, enter:

  xattrschema show *attr_name*

Examples — Displaying xattrschema Information

- The following example lists the names of all extended attributes in the registry prefixed by the cell name:

```
dcecp> xattrschema catalog /.:/sec/xattrschema
/.../dresden/sec/xattrschema/pre_auth_req
/.../dresden/sec/xattrschema/pwd_val_type
/.../dresden/sec/xattrschema/pwd_mgmt_binding
/.../dresden/sec/xattrschema/X500_DN
/.../dresden/sec/xattrschema/X500_DSA_Admin
/.../dresden/sec/xattrschema/disable_time_interval
/.../dresden/sec/xattrschema/max_invalid_attempts
/.../dresden/sec/xattrschema/passwd_override
/.../dresden/sec/xattrschema/test_integer
dcecp>
```

- The following example lists the attributes of the extended registry attribute named **test_integer**:

```
dcecp> xattrschema show /.:/sec/xattrschema/test_integer
{aclmgr {principal {{query r} {update r} {test r} {delete r}}}}
{annotation {test_integer: encoding type integer}}
{applydefs yes}
{encoding integer}
{intercell reject}
{multivalued yes}
{reserved no}
{scope {}}
{trigbind {none {}}}
{trigtype none}
{unique no}
{uuid 5f439154-2af1-11cd-8ec3-080009353559}
dcecp>
```

Enabling and Disabling the Audit Logging Service

Use the DCE control program to enable or disable the audit record logging service of the Audit daemon. The **aud enable** command enables the logging service, and the **aud disable** command disables it.

You may want to disable the logging service when the audit trail file becomes too large, and then enable it again after the audit trail has been backed up and rewound (using the **aud rewind** command).

Using the enable or disable commands enable or disable audit record logging to the central audit trail file. Applications such as the security server and the time server use their own audit trail files and are not affected by use of enable or disable.

The **aud stop** command stops the Audit daemon.

Ensuring Consistent Local Files

Introduction — Ensuring Consistent Local Files

The **passwd_export** command makes the standard /**etc**/**passwd** and/**etc**/**group** files on the local machine consistent with the registry database. Run the **passwd_export** command on a regular but staggered basis, preferably as part of **cron** processing. If **passwd_export** succeeds in creating the new password and group files, it saves the current files as backups that are named **passwd.bak** and **group.bak**. If it fails, it leaves the current files as is.

More Detailed Information — Ensuring Consistent Local Files

The **passwd_export** command has the following format:

passwd_export [-l][-g *group1, group2, ...***][-u** *user1, user2, ...***][-n][-f]**
[-d *directory_name***][-x][-m** *max_entries***][-v][-s][-h[elp]]**

where:

–l Specifies operations are to be performed only on users in the local /**etc**/**passwd** file.

–g *group1, ...* Specifies the group on which to perform the operation. If this option is not specified, all groups are included.

–u *user1, ...* Specifies the users (or principals) on which to perform the operation. If this option is not specified, all users (or principals) are included.

–n Specifies that **passwd_override** and **group_override** entries should be ignored. Without this flag, **passwd_export** applies the override entries to the local password and group files it creates.

–f Forces the update by overriding the **dce_export** attribute in the /**etc**/**security**/**user** file.

–d *directory_name*
Specifies the name of a directory in which to store the password, group, organization files that are created by **passwd_export**. If you do not enter this option, the files are stored by default in the /**etc** directory on the local node. For example, to store the files in the directory that is called /**etc**/**locals**, enter the command as follows:

```
$ dceshared/bin/passwd_export -d/etc/locals
```

–x Prohibits the creation of entries for users with password or group overrides (on the local machine) that specify **OMIT** as their encrypted password. Use the **–x** option to exclude omitted users or groups from the password and group files that are created by **passwd_export**. To omit a user, you must create an override entry for the user and enter the word **OMIT** as the user's password field entry. Omitted users are unable to log into the local machine. See “Omitting Users from Local Password Files” under “More Detailed Information — Overriding Entries” on page 7-95. To omit a group, create an override entry for the group and enter the word **OMIT** as the group's password field entry.

–m *max_entries*
Sets the maximum number of registry entries that can be stored in the local files.

–v Runs in verbose mode.

–s Sorts the entries in the local files by UNIX number. If this option is not specified, the entries are in the order in which they are retrieved from the registry.

–h[elp] Displays help information.

Establishing Trust Relationships

Introduction — Establishing Trust Relationships

Use the **registry connect** command to establish direct trust and transitive trust relationships. Note that, although you can create a direct trust relationship between any two cells, you can create a transitive trust relationship only for those cells connected by a transitive trust path. This command creates two special accounts — one in your cell's registry to represent the foreign cell, another in the foreign cell's registry to represent your cell. The command creates the accounts' principals at the same time. After the trust relationship is established, users in the foreign cell can log into accounts in the local cell and vice versa. You control foreign principals' access to specific objects with ACL entries, just as you do for principals in the local cell.

More Detailed Information — Establishing Trust Relationships

When the accounts are created, the **registry connect** command performs two tasks of which you should be aware. First, it automatically generates a single password that is shared by both accounts. This means that users who log into a cell with which their cell has a trust relationship are seen as the same principal and share the same password. Second, the **registry modify** command generates one UNIX number to represent *all* principals from the foreign cell. This shared UNIX number helps prevent collision between the UNIX numbers of local and foreign principals when objects on a local machine are accessed.

Within the registry and for the purposes of network access, principals are identified by a UUID that represents their fully qualified names (for example, /**.../dresden.com**/**dce**/**users**/**mahler** for the principal

mahler). However, the local operating system on a local machine identifies principals by UNIX number. Because UNIX numbers are not required to be unique across cells, it is possible for two principals from different cells to have the same UNIX number. Thus, a foreign principal that is accessing files in the local cell could have the same UNIX number as the local principal and be seen by the local system as the owner of the local user's files on the local machine.

Creating a UNIX number that is applied to every principal from a given cell that accesses the local cell prevents this from occurring. However, you need to be aware that, because the foreign users all have the same UNIX number, the very feature that prevents them from accessing the local user's files allows them to access each other's files. Because each user from the same foreign cell is seen as the same user, every file on the local machine that is owned by a foreign user can be accessed by every other foreign user from the same foreign cell.

Restrictions — Establishing Trust Relationships

To prevent the widespread proliferation of trust relationships that could result in unwieldy administrative burdens and weakened security, the Security Service imposes the following three rules on transitive trust relationships:

1. Any number of descendent cells can be traversed by a transitive trust relationship, and any number of ancestor cells can be traversed by a transitive trust relationship.
2. No more than one direct trust peer relationship can be traversed by a transitive trust relationship. (A direct trust peer relationship is a direct trust relationship between cells that are neither ancestors nor descendants of each other in the naming hierarchy.)
3. After a hierarchical trust relationship traverses a direct trust ancestor and an optional direct trust peer, it cannot traverse to an ancestor of the peer cell. In other words, after a transitive trust path goes up and across, it cannot go up.

The ramifications of these rules are explained in “Examples — Establishing Trust Relationships” on page 7-77.

Rule 1 Any number of descendent cells can be traversed in a hierarchical trust relationship and any number of ancestor cells can be traversed by a transitive trust relationship.

Rule 2 No more than one direct trust peer relationship can be traversed by a transitive trust relationship.

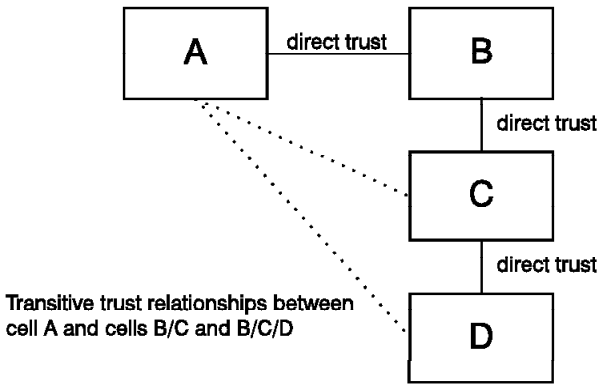
Rule 3 After a hierarchical trust relationship traverses a direct trust ancestor and a direct trust peer, it cannot traverse to an ancestor of the cell.

Note that a principal accessing a foreign cell through transitive trust relationships is not authenticated by each cell transited in the trust path, but only by the target cell itself. The Authentication Service in a transited cell simply gives the principal a ticket to the next cell in the path, stamping the ticket with the hierarchical name of the transited cell, until the principal acquires a ticket to the target cell.

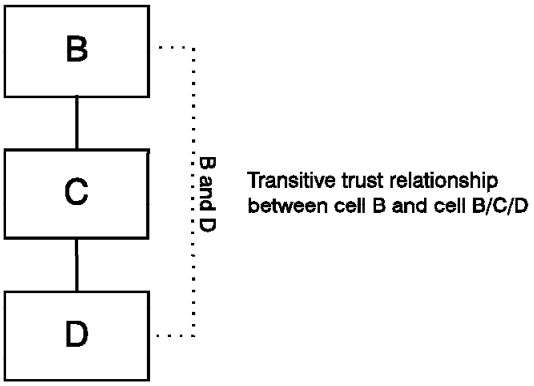
To determine whether to give a principal a ticket to the next cell in a transitive trust path, the Authentication Service in each transited cell examines the ticket and compares the last cell transited to the next cell in the path and applies the rules of transitive trust described in this section. If the next cell to be transited is consistent with a valid transitive trust path, the Authentication Service gives the principal a ticket to the next cell; otherwise, the Authentication Service refuses to issue a ticket.

Examples — Establishing Trust Relationships

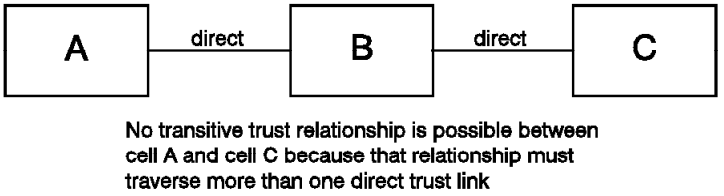
- In the following figure, peer Cells A and B have a direct trust relationship. Cell A has a transitive trust relationship with Cells B/C and B/C/D.



The previous configuration also makes possible the following transitive trust relationship between Cell B and Cell B/C/D as shown in the following figure.

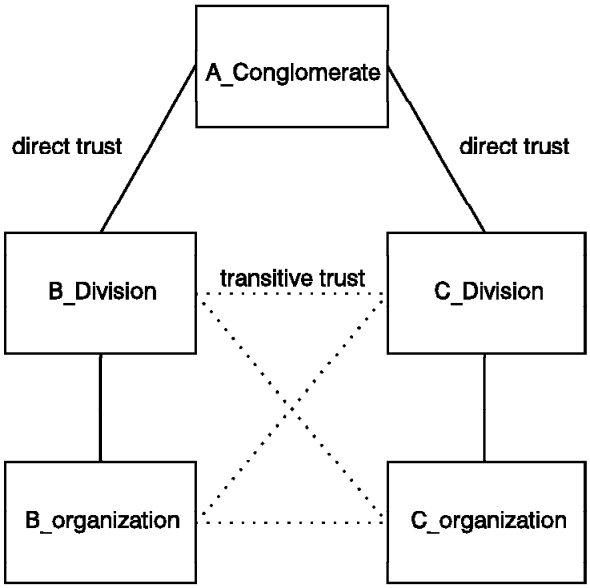


- In the following figure, cells A, B, and C are peer cells. Cell A has a direct trust peer relationship with Cell B, and Cell B has a direct trust peer relationship with Cell C. Cell A does not have a transitive trust relationship with cell C because to do so would traverse more than one direct trust peer relationship (A to B and B to C).



Note that it is not required to traverse a direct trust peer relationship to have a transitive trust relationship. In the following figure, no direct trust peer relationships are traversed. In the figure, a transitive trust relationship exists:

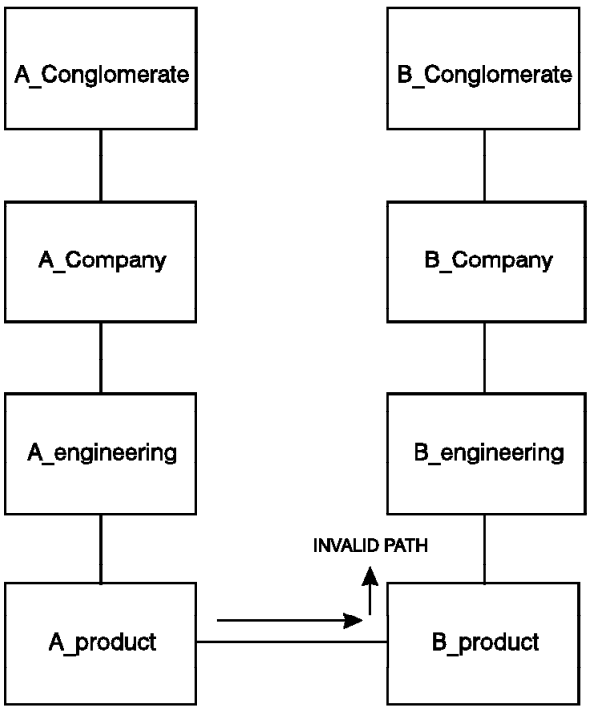
- Between B_Division and C_Division and C_organization and
- Between C_Division and B_Division and B_organization.



In this figure a transitive trust relationship exists:

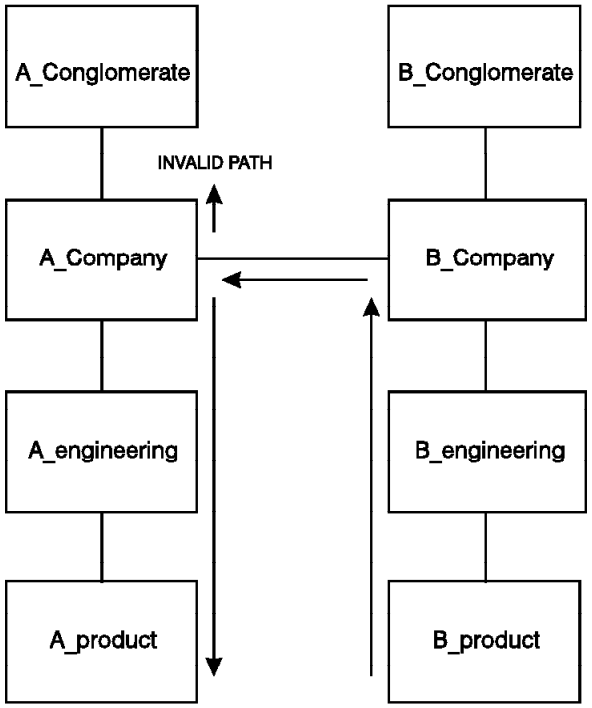
- Between B_Division and C_Division and C_organization and
- Between C_Division and B_Division and B_organization.

- Consider the following figure. The A_Conglomerate cell hierarchy and the B_Conglomerate cell are connected by direct trust relationships. Additionally there is a direct trust relationship between A_product in the A_Conglomerate hierarchy and B_product in the B_Conglomerate hierarchy. In this configuration, no transitive trust relationships are possible because they cannot traverse to an ancestor after traversing a direct trust peer.



The type of trust relationship shown in the preceding figure may be used by two companies that have a very limited agreement to cooperate on product development.

The following figure shows another transitive trust path.



In the path, the B_product cell has a transitive trust path up to its ancestor, B_Company and from B_Company to the A_Company. But, from the A_company, the transitive trust path cannot continue up to A_Company's ancestor, although it can continue down to A_Company's descendants. Because this transitive trust relationship has traversed up to a trust ancestor (B_Company) and across to a trust peer (A_Company), it cannot then continue by going up to A_Company's ancestor (the A_Conglomerate). This type of relationship can be used by two companies that have decided to combine operations at a very high level.

Extended Security Attributes for Principals

Introduction — Extended Security Attributes for Principals

You can attach Extended Registry Attribute (ERA) instances to principals to manage several aspects of DCE login and password security. ERAs are available to control:

- The level of authentication security required for principal login requests,
- Handling of invalid logons,
- Strength of principals' passwords (as well as generation of passwords for principals), and
- Handling of login attempts by principals with expired passwords.

These ERAs are introduced and explained in the following sections. See “Creating and Using Extended Registry Attributes” on page 1-49 for information on how to use **dcecp** to attach these ERAs to principals. See “Maintaining Policies and Properties” on page 1-58 for a description of DCE standard policy and procedures which enable you to control password characteristics.

More Detailed Information — Extended Security Attributes for Principals

DCE for AIX 2.1 Authentication

With the addition of user preauthentication, DCE for AIX 2.1 authentication addresses certain security deficiencies in the Kerberos V5 authentication protocols, used as the basis for the DCE authentication protocol in versions previous to DCE for AIX 2.1. These deficiencies result from:

- The Security server responding to client login requests without verifying that the user knows the password, and
- The use of user passwords, which are notoriously weak, to encrypt plain text data that is then sent across the network.

These practices are subject to attacks in which the attacker obtains network transmissions and proceeds to attack them offline to elicit user's passwords. These kinds of attacks, if successful, can compromise the security of a DCE cell (and of all other cells in a trust relationship with that cell).

DCE for AIX 2.1 reduces the likelihood of such attacks succeeding by providing:

- Preauthorization of principals making login requests (that is, by having the Security Service verify the identity of the requestor before responding to the request), and
- The use of strong keys to encrypt all network transmissions involving validation between security clients and servers.

There are three levels of authentication, ranging from most to least secure, and representing decreasingly strict preauthentication protocols. By attaching an instance of the *pre_auth_req* era (described in the following section) to the principal, administrators can control the minimum level of preauthentication that the security server accepts when authenticating the principal.

The preauthentication protocols are:

- The third-party protocol provides the highest level of security. No lesser level of preauthentication should be specified for any principal unless there is a compelling reason to do so (see the note on *cell_admin* in the next item). DCE for AIX 2.1 clients always construct authentication requests using this protocol, except in cases where they are unable to do so because the machine session key, which is required to construct third-party requests, is unavailable (for example, at cell startup or when the **secval** process is down).
- The timestamps protocol provides an intermediate level of security. Timestamps preauthentication should be specified only for principals (such as cell administrators and noninteractive principals) who must be able to operate when the client is unable to construct a third-party authentication request as previously described.

  In these cases, the client constructs and forwards a timestamps login request. In particular, the cell administrator *must* have timestamps login capability because **cell_admin** must be able to log in to set up the initial machine key during initial configuration of the cell.
- The DCE for AIX Version 2.1 (Kerberos V5) protocol is used to authenticate pre-DCE for AIX 2.1 clients only and provides no preauthentication security.

Managing User Authentication

You manage preauthentication for a given principal by attaching an instance of the *pre_auth_req* ERA to the principal and specifying a value to indicate the lowest level protocol the Security Service should accept for the principal, as follows:

0 (**NONE**) Specifies that the Security Service should accept, from this principal, login requests using any of the three protocols (including the pre-DCE for AIX 2.1 protocol). This is the least secure level and is provided only in order to enable DCE for AIX 2.1 servers to accept login requests from pre-DCE for AIX 2.1 clients. It is most vulnerable to the type of attack described above.

Warning: Failing to attach an instance of the *pre-auth_req* ERA to a principal is equivalent to specifying **0** (NONE).

1 (**PADATA-ENC-TIMESTAMPS**) Specifies that the Security Service should accept, from this principal, login requests using either the timestamp or third-party protocol. The timestamp protocol protects against attackers masquerading as a security client and attacking replies from the Authentication Service, but is still vulnerable to attacks by processes capable of monitoring the network.

2 (**PADATA-ENC-THIRD-PARTY**) Specifies that the only login requests the Security Service will accept from this principal are those using the third-party protocol. This is the highest level of DCE preauthentication and provides the most protection against the attacks described above. With third-party preauthentication, all authentication data sent over the network is encrypted using a strong random key known only to the local machine principal and the Security Service.

When the Authentication Service receives a login request for a principal, it always attempts to respond using the same protocol as the request, unless the *pre_auth_req* ERA value for that principal forbids it to do so.

For complete information on the details of DCE authentication (including the operation of the preauthentication protocols), see the *DCE for AIX Application Development Guide — Core Components*.

Preauthentication Interoperability Between DCE Versions

The following table describes how login requests are handled when DCE for AIX 2.1 client/servers interoperate with pre-DCE for AIX 2.1 clients/servers in a single cell.

Table 7-5. DCE for AIX 2.1/Pre-DCE for AIX 2.1 Authentication Interoperation

Login Request Type	Pre-DCE for AIX 2.1 Server Response	DCE for AIX 2.1 Server Response
DCE for AIX 2.1		
From pre-DCE for AIX 2.1 clients only.	No preauthentication. Returns DCE for AIX 2.1 (unpreauthenticated) response.	Support preauthentication. Checks for *pre_auth_req* ERA instance: • If no ERA exists, or existing ERA has value=0 (NONE), returns DCE for AIX 2.1 (unpreauthenticated) response. • Otherwise, rejects the login response.
TIMESTAMPS		
From DCE for AIX 2.1 clients only.	No preauthentication. Server ignores preauthentication data in request and returns pre-DCE for AIX 2.1 (unpreauthenticated) response.	Support preauthentication. Checks for *pre_auth_req* ERA instance: • If no ERA exists or existing ERA has value=0 (NONE) or value=1 (PADATA-ENC-TIMESTAMPS), returns DCE for AIX 2.1 TIMESTAMPS response. • If existing ERA has value=2 (PADATA-ENC-THIRD-PARTY), rejects the login request.
THIRD-PARTY		
From DCE for AIX 2.1 clients only.	No preauthentication. Server ignores preauthentication data in request and returns pre-DCE for AIX 2.1 (unpreauthenticated) response.	Supports preauthentication. Returns DCE for AIX 2.1 THIRD-PARTY response.

Managing Invalid Login Handling

The Security Server is able to detect and track invalid login attempts for a given principal. This makes it possible for administrators to limit the possible impact of password guessing attacks by:

- Setting a limit to the number of successive invalid login attempts before the principal's account is disabled (a successful login resets the counter) and
- Specifying the period of time the principal's account will be disabled when that limit is reached.

You do this by attaching instances of two ERAs (*max_invalid_attempts* and *disable_time_interval*) to the principal. Specify values for these ERAs as follows:

max_invalid_attempts Specify an integer indicating the number of successive invalid login attempts the Security Server should accept before marking the principal's account as disabled.

disable_time_interval Specify an integer indicating the number of seconds the principal's account should be disabled from login attempts.

Note: At DCE for AIX 2.1, the invalid login handling functionality accurately tracks login activity in a cell with one master and no replicas but does not keep accurate counts in replicated cells. This is because:

- Login attempts in a replicated cell are randomly assigned to either a master or replica.
- There is, at present, no mechanism for replicas to communicate to the master and, therefore, no way for the master to maintain an accurate count.

Managing Password Expiration

By default, the DCE Security server disables logons for principals whose passwords have expired. There may be cases where you would prefer this not to happen; for example, you probably don't want *cell_admin* to be locked out of the cell because of an expired password.

You can manage password expiration checking for a given principal by attaching an instance of the *passwd_override* ERA to the principal and specifying one of the following values:

0 (**NONE**) Specifies that password expiration checking for the principal should not be overridden (that is, the principal should not be permitted to log in with an expired password). Specifying **0** (NONE) is equivalent to not attaching an ERA instance to the principal.

1 (**OVERRIDE**) Specifies that password expiration checking for the principal should be overridden (that is, the principal should be permitted to log in with an expired password).

Examples — Extended Security Attributes for Principals

- The following is an example of a **dcecp** command to create a principal and attach a *pre_auth_req* ERA specifying third-party preauthentication.

```
dcecp> principal create smitty -attribute {pre_auth_req 2}
dcecp>
```

- The following is an example of a **dcecp** command to create a principal and attach *max_invalid_attempts* and *disable_time_interval* ERAs:

```
dcecp> principal create smitty -attribute {{max_invalid_attempts 7}
                                {disable_time_interval 60}
dcecp>
```

- The following is an example of a **dcecp** command to create a principal and attach *pwd_val_type* and *pwd_mgmt_binding* ERAs:

```
dcecp> principal create smitty -attribute {{pwd_val_type 2}
       {pwd_mgmt_binding {dce/.:/pwd_strength pktprivacy secret name}
       {/.:pwd_mgmt/pwd_strength}}}
dcecp>
```

- The following is an example of a **dcecp** command to create a principal and attach the *passwd_override* ERA:

```
dcecp> principal create smitty -attribute {passwd_override 1}
dcecp>
```

Forcibly Deleting a Slave Replica

Introduction — Forcibly Deleting Slave Replicas

The procedure described in this section explains how to forcibly delete a slave replica. Use this drastic method only when the ordinary method of deletion described in “Changing the Master Replica Site” on page 7-37 fails.

More Detailed Information — Forcibly Deleting Slave Replicas

To forcibly delete a slave replica, use the **dcecp registry delete** command with the **–force** option. This command deletes the slave replica from the master's replica list. The master replica then propagates the delete request to the other replicas. Since this operation never communicates with the deleted replica, use the **–force** option only when the replica dies and cannot be restarted. If a forcibly deleted replica continues operation, use the **registry delete** command with the **–only** option to stop the server and delete its database. You can also simply stop **secd** (by using the **dcecp registry stop** command) and delete or rename its database.

To forcibly delete a registry replica, issue the **registry delete** command with the **–force** option supplying the name of the registry to delete as an argument.

If the default replica is not the master, **dcecp** automatically binds to the master.

If a forcibly deleted replica continues operation, use the **registry delete** command with the **–only** option to stop the server and delete its database. When you use the **registry delete –only** command, you must enter the name of the replica that you want to stop.

Alternatively, you can simply stop **secd** (by using the **dcecp registry stop** command) and destroy the replica by deleting or renaming its database.

Examples — Forcibly Deleting Slave Replicas

- The sample below deletes the replica at **/.../giverny.com/subsys/dce/sec/lit_server_2**.

```
dcecp> registry delete /.../giverny.com/subsys/dce/sec/lit_server_2 -force
dcecp>
```

- The following example shows the **registry delete –only** command used to delete the replica at **/.../giverny.com/subsys/dce/sec/lit_server_2**.

```
dcecp> registry delete /.../giverny.com/subsys/dce/sec/lit_server_2 -only
dcecp>
```

Handling Network Address Changes

Introduction — Handling Network Address Changes

When **secd** starts, primary and secondary replicas can detect address changes and can perform the necessary updates to the primary's replica list and to the cell namespace. Generally all that is required on your part to handle network address changes is to update the **pe_site** file. However, if the network address of the primary and a secondary replica change simultaneously, your intervention is required. This section describes how to update the **pe_site** file and how to handle simultaneous address changes.

More Detailed Information — Handling Network Address Changes

Updating the pe_site File

Whenever the primary or secondary replica's network address changes, you must update the **/opt/dcelocal/etc/security/pe_site** file on that host before restarting **secd**. This file, which exists on each machine in the cell, is required for binding by the Security service to itself. For the primary replica, the file contains the cell name and the name of the primary. For secondary replicas, the file contains the cell name, the name of the primary replica, and the name of the replica itself.

If the primary replica address changes, update the **pe_site** file on every node in the cell that runs a Security server (including the primary) with the new address for the primary. If a secondary address changes, update only that secondary replica's **pe_site** file to reflect its changed address.

Handling Simultaneous Address Changes

If an address change occurs simultaneously for the primary replica and a secondary replica, the primary and secondary will not be able to reach each other while both are trying to notify the other of the changed address. To avoid this problem, make sure the address change of one replica (either primary or secondary) is propagated to all replicas before the other address is changed. Make one address change. Then, use the **dcecp registry show –replica** command to view the replica list at both the primary site and the secondary replica site. When the new address is displayed, on both replica lists, it is safe to proceed with the next network address change.

If you are unable to prevent simultaneous network address changes for the primary and a secondary, the only way to restore communication between the primary and secondary is to delete the secondary, then recreate it. Delete the secondary using one of the following methods, depending on your circumstances:

- If you anticipate a simultaneous address change, while the primary and secondary are still communicating, use the **dcecp designate** command to bind to the primary and then the **dcecp registry delete** command to delete the secondary replica.
- If the **secd** is running at the primary and secondary sites, but the primary and secondary are not communicating, first use the **dcecp designate** command to bind to the secondary and then the **registry delete –only** command to destroy the secondary. Then use **dcecp designate** to bind to the primary and registry **delete –force** command to remove the replica list entry for the secondary.
- If **secd** is not running at the secondary site or if you are unable to bind to the secondary site, use the procedure for recreating a replica described in “Troubleshooting Procedures” on page 1-58.

How Identities Represented by Accounts are Authenticated

Introduction — Identity Authentication

When principals log into DCE, the security client uses the password they supply (or that is supplied for them in the case of a server or machine principal) to derive the principal's authentication key. A copy of the authentication key, based on the password supplied when the principal's account was created or changed, is also stored in the registry database. It is, therefore, available to the Authentication Service.

This key is used by the Authentication Service to authenticate the principal (that is, to guarantee the principal's identity) as follows:

1. The security client:
   a. Queries the user for the password and uses it to derive the principal's authentication key,
   b. Prepares a login request (part of which is encrypted using the authentication key), and
   c. Forwards the request to the Authentication Service.
2. The Authentication Service:
   a. Receives the login request,
   b. Obtains the Registry's copy of the principal's authentication key, and
   c. Attempts to decrypt the login request using this key.

If the decryption succeeds, the keys are the same; the principal is, therefore, authenticated and the login is successful.

If the decryption fails, the password supplied by the principal and used by the security client to derive its version of the principal's authentication key, is invalid (that is, different from the password used to derive the registry's copy of the principal's authentication key) and login is denied.

This is a very general introductory description. See the *DCE for AIX Application Development Guide — Core Components* for a detailed discussion of principal authentication.

More Detailed Information — Identity Authentication

Privilege Attributes

After a principal is authenticated, the Security Service helps obtain the principal's privilege attributes. Privilege attributes consist of UUIDs that represent the principal's network identity, the groups in which the principal is a member, and any extended attributes associated with the principal. They are used to determine their rights to those objects when principals request access to objects. Privilege attributes that are provided by the Security service are authenticated. Authenticated privileges are accepted by network services. Unauthenticated privilege attributes may not be accepted. This means that the kinds of access to DCE objects that principals are allowed can differ, depending on whether or not a principal's privilege attributes are authenticated. (The DCE Access Control Lists, which are used to control access to DCE objects based on a principal's privilege attributes, are described in *DCE for AIX Administration Command Reference*.)

Ticket-Granting Tickets and Tickets to Services

A ticket-granting ticket allows a principal to request and receive tickets to DCE services, such as to a Distributed File System server to read a file. The tickets that let principals access DCE services are called *service tickets*.

Both ticket-granting tickets and service tickets have lifetimes that are determined by the settings for individual accounts and registry policies and properties. When a principal's ticket-granting ticket expires, the principal is no longer considered an authenticated user. An authenticated principal's access to objects other than those on the local machine is severely curtailed and the principal's ability to use DCE services becomes extremely limited. To remedy this, the principal must reauthenticate by running the **kinit** command (described in the *DCE for AIX Administration Command Reference*) or by logging out and then into DCE again.

Note: The **kinit** command fails if you are using AIX 1.3 clients with AIX 2.1 servers. You must install PTF U438707 on your DCE 1.3 client systems to enable **kinit** to run successfully.

If you flag an account as able to renew service tickets, the principal's service tickets are renewed automatically by the Authentication Service, requiring no action on the principal's part. Note, however, that the lifetime allocated to a service ticket can never exceed the time remaining on the principal's TGT.

Displaying Privilege Attributes and Tickets

DCE cell administrators can use the **klist** command to display a principal's current tickets and privilege attributes. The **klist** command displays three types of information — privilege attributes, expiration information, and service ticket information. DCE users can also run **klist** to display their current and expired tickets. The **klist** command is described in the *DCE for AIX Administration Command Reference*.

The First Part of the **klist** *Display — Privilege Attributes*

The **klist** command displays a principal's privilege attributes. This display first lists the fully qualified principal name, followed by the UUIDs and names of the cell, the principal name (without the cell name and DCE global identifier), and all the groups of which the principals is a member. A sample of this section of the **klist** display follows:

```
DCE Identity Information:
  Global Principal: /.../dresden.com/music/mozart
  Cell:             5ad96550-80c4-11ca-b26c-08001e039431 /.../dresden.com
  Principal:        00000066-80c5-11ca-b600-08001e039431 music/mozart
  Group:            00000003-80c4-11ca-b201-08001e039431 composers
```

The Second Part of the klist *Display — Expiration Dates and Times*

The second part of the **klist** display shows the dates and time that the principal's ticket-granting ticket, account, and password expire:

- The first line shows the date and time the ticket-granting ticket expires. Before this happens, the principal should reinitialize it by running **kinit** or logging in again to DCE.
- The second line shows when the principal's account expires. If the account expires, the principal will be unable to log into DCE. To remedy this, DCE administrators must change the principal's account expiration date in the registry.
- The third line shows the date the principal's password expires. the **dcecp** command. If the password expires, the principal will be unable to log into DCE. To remedy this, DCE administrators must change the principal's password in the registry.

A sample of the second part of the **klist** display follows:

```
Identity Info Expires:  91/10/03:12:07:18
Account Expires:        91/12/31:12:00:00
Passwd Expires:         91/10/31:12:00:00
```

The Third Part of the klist Display — Tickets

The third and final part of the **klist** display shows the principal's ticket information and the name of the principal's ticket cache. The first three tickets labeled `Server` in the following display are the tickets used after the principal logged in and obtained privilege attributes. The display for all principals has these entries.

The remaining tickets labeled `Client` show the principal's ticket-granting ticket and service tickets. In the listing for each ticket after the word `Client`, the display shows the name of the privilege server, a server that grants privilege attributes after the principal's identity has been authenticated by the DCE Security Service. The name of the server to which the principal has tickets is shown after the `Server` entry, and the dates and times these tickets are valid are shown on the following line. For example, in the following sample display, the last line shows that the principal has a ticket to the server named **file_server**. The lifetime of this ticket is from 1:24 and 2 seconds p.m. on 10/2/91 to 12:07 and 18 seconds p.m. on 10/3/91. (The time is shown in 24-hour format.)

```
Kerberos Ticket Information:
Ticket cache: /tmp/dcecred_17a80000
Default principal: music/mozart@dresden.com
Server: krbtgt/dresden@dresden.com
   valid 91/10/02:12:07:18 to 91/10/03:12:07:18
Server:dce/rgy@dresden.com
   valid 91/10/02:12:07:20 to 91/10/03:12:07:18
Server:dce/ptgt@dresden.com
   valid 91/10/02:12:07:49 to 91/10/03:12:07:18
Client:dce/ptgt@dresden       Server:krbtgt/dresden@dresden.com
   valid 91/10/02:12:07:50 to 91/10/03:12:07:18
Client:dce/ptgt@dresden.com   Server:dce/rgy@dresden.com
   valid 91/10/02:12:07:53 to 91/10/03:12:07:18
Client:dce/ptgt@dresden.com   Server:file_server@dresden.com
   valid 91/10/02:13:24:02 to 91/10/03:12:07:18
```

Destroying a Principal's Tickets

Use the **kdestroy** command to invalidate the tickets that a principal has acquired. When the principal logs out, the principal's tickets are not destroyed; they remain valid until they expire. DCE users may want to use **kdestroy** just before they log out to ensure that no valid tickets remain. However, if the principal has the kernel-resident ticket cache, the principal's tickets are destroyed when the principal's last process terminates. This means that it is generally not necessary to run **kdestroy** at logout time.

The **kdestroy** command is described in the *DCE for AIX Administration Command Reference*.

How the Registry Database is Stored

Each Security server maintains a working copy of the registry database in virtual memory and a permanent copy on disk. All reads and updates operate on the copy that is in virtual memory. The servers use the copy that is on disk to initialize the copy in virtual memory when they start up. An atomic update log is used to guarantee the database state in the event of server failure.

Figure 7-5 on page 7-89 shows the server, disk, and virtual memory copies of the registry database.

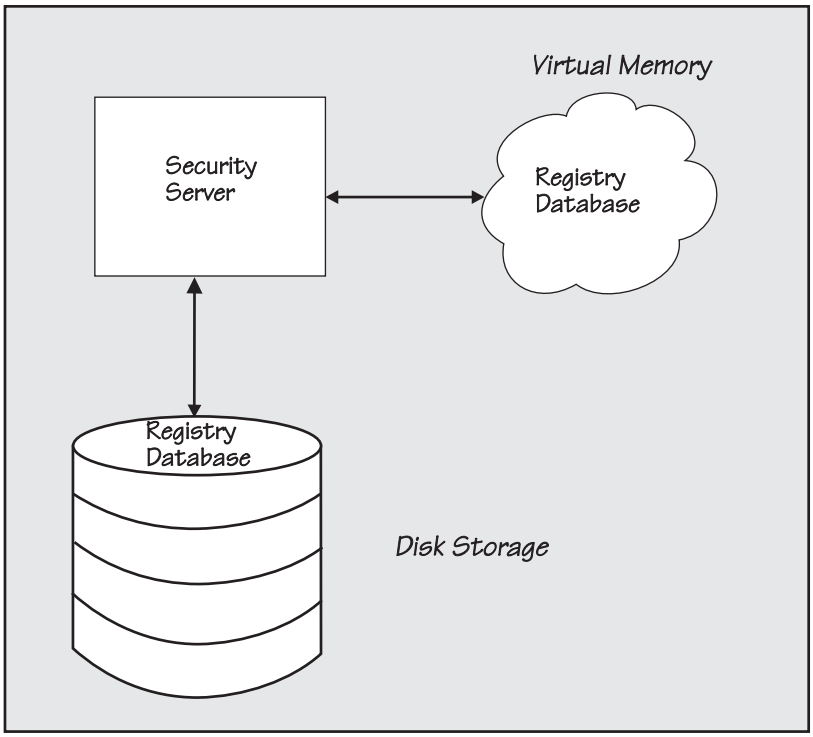


Figure 7-5. Disk Memory and Virtual Memory Copies of the Registry Database

Each Security server periodically saves its entire database from virtual memory to disk. The database is stored in **/opt/dcelocal/var/security/rgy_data**.

Initial Registry ACLs

Introduction — Initial Registry ACLs

When the registry database is created, the **principal**, **group**, and **org** directories and the **policy**, **replist**, and **xattrschema** objects are given initial ACLs. As new objects are created in the registry, they inherit their ACLs from the **principal**, **group**, and **org** directory ACLs. The ACL entry key for those initial ACL entries that require a key is the name of the principal that creates the registry database (supplied to the **sec_create_db** command as the registry creator) or **root** if no name is supplied. See "Setting Up the Registry" on page 7-139, for more information on **sec_create_db** and the registry creator.

More Detailed Information — Initial Registry ACLs

The initial ACLs that are created when the registry database is created are described in the following list. In the list, *rgy_creator* represents the principal that is named as the registry creator.

Objects	Initial Objects (-io)	Initial Containers (-ic)
/.:/sec/principal		

Objects	Initial Objects (-io)	Initial Containers (-ic)
unauthenticated:r----- user:*rgy_creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----	unauthenticated:r-------g user_obj:r---f--ug user:*rgy_creator*:rcDnfmaug group:acct-admin:rcDnfmaug other_obj:r-------g any_other:r--------	unauthenticated:r----- user:*rgy_creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----
/.:/sec/group		
unauthenticated:r----- user:*rgy_creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----	unauthenticated:r-t----- user:*rgy_creator*:rctDnfmM group_obj:r-t----- group:acct-admin:rctDnfmM other_obj:r-t----- any_other:r-t-----	unauthenticated:r----- user:*rgy_creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----
/.:/sec/org		
unauthenticated:r----- user:*rgy_creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----	unauthenticated:r-t----- user:*rgy_creator*:rctDnfmM group:acct-admin:rctDnfmM other_obj:r-t----- any_other:r-t-----	unauthenticated:r----- user:*rgy_creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----
/.:/sec/policy		
unauthenticated:r---- user:*rgy_creator*:rcmas group:acct-admin:rcma- other_obj:r---- any_other:r----		
/.:/sec/replist		
user:*rgy_creator*:cidmA group:acct-admin:cidmA		
/.:/sec/xattrschema		
unauthenticated:r---- user:*rgy_creator*:rcidm other_obj:r---- any_other:r----		

Note: Your platform's configuration tool may update these initial ACLs.

Machine Accounts

All machines have accounts in the registry. Machine accounts, like server accounts, are created by first running the **account create** command to create the account and then running the **keytab add** command to add the server's password to the keytab file. Like server accounts, the password for a machine account in the registry and in the keytab file must match. Principal names in machine accounts must be the same as the machine's name in the cell namespace. See “The CDS Space” on page A-12 and “The Security Space” on page A-27 for more information on names in the cell namespace.

Maintaining Membership Lists

Introduction — Maintaining Membership Lists

Each group or organization has a membership list which identifies the principals that are members of the group or organization. Use the **dcecp group add** command to add members to the membership lists and the **dcecp group remove** command to remove members from the list.

If you delete a member from a group or organization, any accounts for the deleted member that are associated with the group or organization are also deleted. For example, if you delete the principal **mahler** from the group **symphonists**, the account **mahler symphonists classic** is also deleted.

Note: Deleting a principal from a group or organization can affect the principal's rights to objects. This change takes effect only when the principal's ticket-granting ticket is renewed. See “Managing Intercell Naming” on page 1-31 for more information on ticket renewals.

More Detailed Information — Maintaining Membership Lists

Effects of Account Creation on Membership Lists

When you create accounts, the principal for whom the account is created must be a member of the group or organization that is named in the account. For example, if you create the account **mahler symphonists classic**, the principal **mahler** must be a member of the **symphonists** group and the **classic** organization.

The **dcecp** command recognizes this requirement and, if you have the permissions to add to the group or organization, it tries to add the principal to the group and organization. For example, assume that the principal **mahler** is not a member of either the group **symphonists** or the organization **classic**. If you have the proper permissions when you create the account **mahler symphonists classic**, the **account create** command automatically adds **mahler** to the **symphonists** and **classic** membership lists so that you can create the account in one step.

However, if you do not have required permissions to the group, the command fails and displays a message similar to the following:

```
Not authorized to perform operation
```

Adding and Deleting Group Members

In the unusual case where you are changing a host's group name information while the host is logged into a DCE cell, the existing host credentials will become invalid unless you perform extra steps to update the host credentials with the new group name information.

Host credentials are managed by the **secval** process which performs security client functions on a DCE host. Normally, just after the host starts, the **secval** process logs the host into the DCE cell, getting the host credentials and storing them on the host. Deactivate and reactivate the **secval** process to update these credentials after changing the group name information.

Examples — Maintaining Membership Lists

- The following example shows the use of the **dcecp group add** command with the **–member** option to add **machler** to the **symphonists** group and delete **strauss** from the **symphonists** group.

```
dcecp> group add symphonists -member mahler
dcecp> group remove symphonists -member mahler
dcecp>
```

- You can add and remove multiple members with one **group add** or **group remove** command. To do so, enclose the member names in quotes, separated by spaces. For example, to add the principals **bach**, **britten**, and **mahler** to the group **symphonists**, you enter:

```
dcecp> group add symphonists -member {bach britten mahler}
```

- The following example illustrates the **activate** and **deactivate** operations on the remote host **persephone**.

```
dcecp> secval deactivate /.:/hosts/persephone/config/secval
dcecp> secval activate /.:/hosts/persephone/config/secval
dcecp>
```

Maintaining the Local Registry

Introduction — Maintaining the Local Registry

The local registry allows login from that machine if a network registry is not available. The local registry is created automatically the first time that a human or nonhuman user performs a DCE login from the local machine if the network registry server is running. As users log into the machine, their account information is automatically added to the local registry.

More Detailed Information — Maintaining the Local Registry

The following files make up the local registry database:

- **/opt/dcelocal/var/security/lrgy_data** — Contains account information entries
- **/opt/dcelocal/var/security/lrgy_tgts** — Contains ticket-granting ticket entries
- **/opt/dcelocal/var/security/lrgy_lock** — Used by the Security server to lock the registry for read/write operations

You must use the Security service **rgy_edit** command to maintain the local registry. The following subcommands are available:

view	To view all entries in the local registry.
delete *principal_name*	To delete the principal specified by *principal_name* from the registry.
properties	To set and view properties for the local registry.
help	To obtain help information on the local registry.

In addition to the subcommands in the preceding list, the **rgy_edit** command provides one additional subcommand, **purge**, and an additional registry property, called the *registry capacity*, to restrict the number of entries that are in the local registry.

The remainder of this section describes the **purge** subcommand and the registry capacity property. See the **rgy_edit** reference page in the *DCE for AIX Administration Command Reference* for information on the use of the other subcommands.

The Registry Capacity Property

The **rgy_edit properties** command lets you set a limit on the number of entries that the local registry can contain. For example, assume that the registry capacity is set to 10 and that the registry contains 10 entries. If a new user logs in from the local machine, an entry is created for that user. This new entry overwrites the oldest of the original 10 entries. As users log in from the machine for the first time, their newly created registry entry overwrites the oldest entry.

Setting the Capacity and Lifespan Properties

You can set a limit on the number of entries that the local registry can contain. This limit is defined by the **Capacity** attribute of the local registry object. Setting the value of this attribute to a moderate number of entries can keep the local registry from growing too large.

For example, assume that the **Capacity** attribute value for the local registry on a machine is set to 10 and that the local registry contains 10 entries. If a new user logs in from the local machine, an entry is created for that user. This new entry overwrites the oldest of the 10 entries currently existing in the local registry. As users log in from the machine for the first time, their newly created registry entry overwrites the oldest entry.

To set the **Capacity** attribute value for the local registry, use the **rgy_edit properties** command.

Note that, when you first enter the command, it displays the current registry capacity and lifespan. In addition, it shows the number of account entries that are in the local registry and the number of TGT (ticket-granting ticket) entries. There is one TGT entry, which contains the account's credentials, for each account entry.

```
$ rgy_edit -l
rgy_edit=> properties

Local Registry Properties:
  Capacity:   25 entries
  Contains:   6 account entries
  Contains:   6 tgt entries
  Lifespan:   3w

Do you wish to make changes [y/n]? (n) y
Enter local registry capacity: (25) 50
Enter acct lifespan in days or 'forever': (3w) 12w
```

In the preceding example, **12w** specifies a 12-week lifespan. When you enter the lifespan, you can enter:

- **forever** — To designate that the entry will never expire. The entry can be overwritten if it exceeds the registry capacity, however. Entering a 0 (zero) at the `Enter acct lifespan in days or 'forever': (3w)` prompt has the same effect as entering **forever**.
- **A decimal number** — To specify the number of measurement units (weeks, days, hours, minutes, seconds). This number is generally accompanied by a measurement unit, as described next. If you enter only a number without an accompanying measurement unit, the measurement unit defaults to days.
- **w, d, h, m, or s** — To specify the measurement units:

  w for weeks
  d for days
  h for hours
  m for minutes
  s for seconds

Purging Expired Entries

The **rgy_edit purge** subcommand deletes expired entries from the local registry. When the **rgy_edit purge** command is run, the command deletes all entries whose lifespan attribute value indicates are expired.

Examples — Maintaining the Local Registry

- You can use any combination of the measurement units with their accompanying numbers. For example, to set the lifespan to 12 weeks, 8 hours, and 30 seconds, enter the following command:

  ```
  Enter acct lifespan in days or 'forever': (3w) 12w8h30s
  ```

- If you end a string of numbers and measurement units with a number only, the number with no measurement unit defaults to seconds. For example, if you enter

  ```
  Enter acct lifespan in days or 'forever': (3w) 12w30
  ```

  the lifespan is assumed to be 12 weeks and 30 seconds.

- In the preceding example, we reset the local registry's lifespan attribute value to **12w**. With the lifespan set to **12w**, the following **rgy_edit purge** command deletes all local registry entries (account and TGT) that are older than 12 weeks.

  ```
  rgy_edit=> purge

  1 account entries purged
  1 TGT entries purged
  rgy_edit=>
  ```

Modifying and Querying Audit Daemon Attributes

The DCE Audit daemon has two attributes which relates to the audit trail file.

stostrategy Specifies the storage strategy when the size of the audit trail file has reached its limit. You can specify any of the following storage strategy:

- **save** If the specified trail size limit is reached, the Audit daemon saves the current trail file to a new file (renaming it to its original name with a timestamp appended at the end of the name). The Audit daemon then deletes the contents of the original trail file and continues auditing from the beginning of this file. This is the default value for **stostrategy**.
- **wrap** The Audit daemon overwrites the old audit trails.

state Indicates whether the Audit daemon is servicing audit record logging requests from audit clients. The possible values for this attribute are **enabled** (default value) or **disabled**.

You can use the DCE control program to see the value of these settings.

```
dcecp> aud show
{state enabled}
{stostrategy save}
```

Use the **aud modify** command to change these attributes.

Modifying Cross-Cell Authentication Accounts

You can change the account that is created by the **registry connect** command at any time using the standard **dcecp** account operations. However, you should be aware of the following cautions.

- Never set the account's **pwdvalid** attribute to **no** (invalid). For standard accounts setting the attribute to **no** prompts users to change their passwords at the next login. Passwords for cross-cell authentication accounts, however, are shared by the Authentication Services in two cells. If you change one, this synchronization is destroyed and cross-cell communication ends. If you want to

change the passwords that are shared by Authentication Services, you must rerun the **registry connect** command to recreate the accounts and create the properly synchronized passwords.

- Do not delete the accounts or the account's principals unless you are breaking the trust relationship with the cell. If one of the accounts is deleted, you must run the **registry connect** command to recreate both accounts and restore the trust relationship.

Object Creation Quotas

You can assign an object creation quota to each principal. This assignment allows you to control the number of registry objects that can be created by the principal. If you allow users to create their own groups, for example, you can use this quota to limit the total number of groups they can create. The default for the object creation quota is **unlimited**, meaning that no limits are placed on the number of objects the principal can create. A value of **0** (zero) prohibits the principal from creating any registry objects.

Each time a principal creates a registry object, the principal's object creation quota is decremented by 1. When the object creation quota reaches 0, the principal is prohibited from creating registry objects unless you reset the object creation quota to a number other than 0 by using the **dcecp principal modify** command. Note that, when an object created by a principal is deleted, the principal's object creation quota is *not* incremented.

Use the **dcecp principal show** command to view principals' current object creation quotas. This command displays the total number of objects that the principal is allowed to create at the current time (that is, the original quota minus the number of objects created by the principal).

Overriding Entries in the Local /etc/passwd and /etc/group Files

Introduction — Overriding Entries

You can override registry entries for local machines. By using overrides, you can, for example, prevent individuals and groups from logging into a particular machine, establish local root passwords, and tailor local user environments. The override information is in effect for the local machine only and has no effect on the account information stored in the registry.

The override mechanism provides a high level of local autonomy and allows individual users to control their own machines. For example, an administrator who is responsible for a group of machines can use the override facility to restrict access to those machines. The administrator can allow access to specific groups, or the administrator can allow access to everyone except specific groups or principals.

More Detailed Information — Overriding Entries

Override Operation

The **passwd_override** administrative file stored in the local machine's /**opt**/**dcelocal**/**etc** directory contains override information. By using this file, you can enter overrides for the following:

- Passwords
- GECOS Information
- Home Directories
- Login Shells
- Group Memberships
- UNIX IDs for Principals

The override information that you enter is in effect only for the local machine, which is the machine on which the **passwd_override** file is stored. When a user logs into a machine with an override file, any information for the user's account in the override file replaces the pertinent information obtained from the registry.

The passwd_override File Format

Entries in the **passwd_override** file have the following format:

principal:passwd:UNIX_id:grp_id:GECOS:home_dir:shell

where:

principal Is a principal name that identifies the account to which the overrides apply.

passwd Is the UNIX encrypted password. You can specify an override in this field. If you do, the password you enter here will be in effect for this local machine only. For no override, leave the field empty.

You can also specify **OMIT** in the *passwd* field to disallow DCE login on the local machine. The use of **OMIT** in conjunction with an option to the **passwd_export** command prevents the inclusion of this user in the password file created by **passwd_export**.

UNIX_id Is the UNIX user ID. You must enter an *UNIX_id* or principal name to identify the account. Enter a UNIX ID when you want to apply the overrides to all accounts for a principal (those identified by the principal's primary name and any aliases). UNIX IDs are especially useful for overrides to **root**. If, for example, **root** has an alias of **wizard**, an override keyed by principal name applies only when **root** logs in as **root**, not when **root** logs in as **wizard**. If the override is keyed by **root**'s UNIX ID, it applies when **root** logs in as **root**, **wizard**, or any other alias.

Note that if you supply a principal name and a UNIX ID, the UNIX ID is used only to override the network UNIX ID, not a key field for lookup.

grp_id Is the UNIX ID that specifies the local identity of the group to which the override applies. Enter a group UNIX ID when you want the override to apply to all members of the group.

In the absence of a *principal* or *UNIX_id* key in the entry, *grp_id* is used as a key field. If a *principal* or *UNIX_id* key is present in the entry, *grp_id* is used as an override value.

GECOS Represents the account's GECOS field. You can specify an override in this field. To keep it unchanged, leave it empty.

home_dir Is the account's home directory. You can specify an override in this field. To keep it unchanged, leave it empty.

login_shell Represents the account's login shell. You can specify an override in this field. To keep it unchanged, leave it empty.

The *principal, UNIX_id*, and *grp_id* fields are key fields. You must enter one of them to identify the principal or group to which the overrides apply. The key field is used to perform a lookup in the override file. The lookup is performed in order — first by principal name, then UNIX ID, and finally group ID. If you specify more than one of these fields in an override entry, the principal name is used as the lookup key and the UNIX ID and group ID are used as overrides.

Using OMIT

If you enter either the word OMIT or another invalid password string (such as an * or “NO GOOD”) in the *passwd* field, the user (or set of users) will be unable to login to the local machine. If you specify OMIT and run **passwd_export** with the **–x** option, the named user (or set of users) will not appear in the **passwd** file produced by **passwd_export**.

The group_override Format

Entries in the **group_override** file have the following format:

group_name:passwd:group_id:members

where:

group_name
: Is a key field that contains the name that identifies the group to which the override applies.

passwd
: Is the UNIX encrypted password. You can specify an override in this field. If you do, the password you enter here will be in effect for this local machine only. For no override, leave the field empty.

  You can also specify **OMIT** in the *passwd* field to disallow DCE login on the local machine. The use of **OMIT** in conjunction with an option to the **passwd_export** command prevents the inclusion of this user in the password file created by **passwd_export**.

group_id
: Represents the UNIX group ID. This field can function as a key field (when no other key fields are entered) or as a field containing an override (when entered in conjunction with *group_name*). This field specifies the local override of the group ID supplied by the network registry server.

members
: Specifies a comma-separated list of members of the group. The contents of this field will override the information in the registry when the **passwd_export** command creates an /**etc**/**group** file. To specify a null membership, as opposed to indicating that no override is required, use an asterisk (*) for this field.

Leaving Fields Blank

If you want to override an item, leave its field blank, separating each blank field with a colon (:). To override a group with a null membership list, enter an asterisk (*) for the *members* field.

Using OMIT

If you specify **OMIT** and run **passwd_export** with the **–x** option, the named group (or set of groups) will not appear in the /**etc**/**group** file produced by **passwd_export**.

On a system with AIX/DCE integrated security features, the **group_override** file can also be used to prevent local access by DCE users. This is done by specifying a *groupname* and an invalid password in a **group_override** entry. Then, any user who is a member of that group is denied local access. For example, the entry

```
temps:OMIT::
```

will prevent any DCE user that belongs to group **temps** from logging onto the local system. It does not matter whether **temps** is the user's primary group or an arbitrary group in the user's group set. Access denial by the **group_override** file is in effect only for integrated BOS functions (not DCE-only functions like **dce_login**) and denial is only on a group name basis. To deny access based on **GID**, use the **passwd_override** file mechanism.

Creating Override File Entries

The local system administrator must create the override files. To create override file entries, edit the **passwd_override** file and supply the override entries. The entry must identify the account (or accounts) to which the override applies by specifying one of the following key fields:

principal_name The name of the specific principal to which to apply the overrides. The override applies only to the account for the principal's primary name. For example, if you specify **mahler** as the principal name in an override entry, the overrides apply only to principal **mahler**'s account and not to any accounts for **mahler**'s aliases.

principal_uid A UNIX ID that identifies the accounts to which to apply the override if *principal_name* is not specified. The override is applied to all accounts for the principal that is identified by *principal_uid*, including any accounts for the principal's aliases. For example, suppose that principal **mahler** has a UNIX ID of **2195**. If you specify **2195** as the key of the entry, the overrides apply to all accounts that are associated with that UNIX ID. Because a principal's primary name and aliases carry the same UNIX ID, this means that the overrides apply to accounts for the principal's primary name and all aliases.

group_uid A UNIX ID that identifies the group to which to apply the overrides if neither *principal_name* nor *principal_uid* are specified. The overrides are applied to all accounts for all principals that are members of the identified group.

The *principal_name* field always acts as the key field and cannot be overridden. If you enter *principal_name*, it identifies the specific account to be overridden. The *principal_uid* or the *group_uid* field can act as the key field or they can act as override fields. Only one of the possible key fields is used as a key for any one entry; the others (if entered) are used as override fields. The *principal_name* field takes precedence, followed by *principal_uid*, and finally *group_uid*.

For example, if you enter *principal_uid* and do not enter *principal_name*, *principal_uid* is used as the key field. If you enter *principal_uid* and *principal_name*, *principal_name* is used as the key field and *principal_uid* is used as an override field.

If you enter the *group_uid* field and do not enter any other key fields (*principal_name* or *principal_uid*), *group_uid* is used as the key field, and the overrides apply to the accounts of all members of the group. If you enter *principal_name* and *group_uid*, *principal_name* is used as the key field, and the group affiliation of the named principal is overridden by the group that is identified by the group UNIX ID.

Leaving passwd_override File Fields Blank

If you do not want to override an item in the **passwd_override** file, leave its field blank, separating each blank field with a colon (:). You must enter one of the key fields, however, to identify the principal or group for which you are creating overrides. For example, an entry to override the home directory for the account identified by **mozart** looks like:

```
mozart:::::/aria/wolfgang:
```

You must enter the colons associated with any blank trailing fields. In the preceding example, a colon is required for the shell field, which is the remaining field after the home directory field.

Specifying Passwords for a Particular Machine

To create an entry in the override file to override passwords on the local machine, manually edit the password entry in the **passwd_override** file on the local machine. The password that you enter when you manually edit the file must be encrypted, but you can copy the encrypted password from the /**etc**/**passwd** file or you can write a program that generates encrypted passwords.

When you override a principal's password, only the principal's local credentials are obtained at login, not the principal's network credentials. Without network credentials, the principal cannot access the network registry and obtain the information that is normally provided at network login. Therefore, you must supply all of this information in the **password_override** file entry. For overrides to passwords, you must enter all of the fields in the override entry, including all key fields.

Preventing Login to a Machine

To prevent users from logging into a machine, create an override entry with an invalid string in the **passwd** field. Because the **passwd** field contains an encrypted password, any character string that is not exactly 13 characters in length can be used as an invalid password. For example, the following entry in the **passwd_override** file supplies **exclude** as a password. This string of less than 13 characters prevents members of the group that is identified by a UNIX ID of **25** from logging in.

```
:exclude::25:::
```

Omitting Users from Local Password Files

An invalid password entry in the **passwd_override** file prohibits users from logging into the machine on which the file exists. However, the invalid entry **OMIT** has a special meaning. Just as with any other invalid password, if you enter **OMIT**, the user cannot log in. Additionally, however, if you maintain the standard /**etc**/**passwd** and /**etc**/**group** files and use the **passwd_export** command to keep these files consistent with the registry database, you can specify that users with a password of **OMIT** be excluded from the /**etc**/**passwd** file. See “Ensuring Consistent Local Files” on page 7-74 for more information on the **passwd_export** command.

Also be aware that, if you have omitted users from the /**etc**/**passwd** file, information about those users is not available to any program that uses the password file and is not DCE-aware.

Specifying a Home Directory and Login Shell for a Machine

To change an account's home directory and login shell for a specific machine, create an override entry with a home directory name and a login shell name. For example, the following entry changes the home directory and login shell for user **mozart**'s account:

```
mozart::::::rondo/mozart:/bin/ksh
```

Overriding a Principal's Group Affiliation

To override a principal's group affiliation, create an override entry that contains the principal's name or UNIX ID as a key and the UNIX ID of the group that is to be used as the override. Use the principal's name as a key to apply the overrides only to the account for the principal's primary name. Use the principal's UNIX ID as a key to apply the overrides to all of the principal's accounts, including any accounts for the principal's aliases. For example, the following entry overrides the group that is normally associated with the account for principal **mozart**:

```
mozart:::356::
```

This override does not apply to any accounts for any of **mozart**'s aliases. To apply the overrides to those accounts, the entry must be keyed by **mozart**'s UNIX ID (**567**), as follows:

```
::567:356::
```

Applying Overrides to All Members of a Group

To apply overrides to all members of a specific group, create an override entry that contains the group's UNIX ID as a key and the items to override. For groups, you can override passwords, GECOS information, home directories, and shells. For example, the following entry makes /**sonata**/**piano** the home directory for all members of the group that is identified by UNIX ID **356**:

```
:::356::/sonata/piano:
```

Be sure not to include the *principal_name* or *principal_uid* key fields. If you do, the principal name or UNIX ID that you supply will be used as a key field, and the group UNIX ID will be used to override that principal's group affiliations.

How passwd_override Handles Multiple Override Entries

When more than one override entry applies to an account, the entry with the most specific account identifier (that is, either a principal UNIX ID, a group UNIX ID, or a principal name) is selected. Principal names are the most specific, followed by the principal UNIX ID and group UNIX ID.

Examples — Overriding Entries

- Assume that the registry account for **bach** specifies a Korn shell at login. Since **bach** normally logs into a machine that can run a Korn shell, this is fine for a majority of situations. However **bach** occasionally works for another department and logs into a machine that cannot run a Korn shell. To accommodate **bach**'s needs, you can create an override file on the machine that cannot run the Korn shell. The override can specify a Bourne login shell. Then, if **bach** logs into the machine that can run a Korn shell, registry data is used and a Korn shell is invoked. When **bach** logs into the machine that cannot run a Korn shell, override data is used and a Bourne shell is invoked instead.
- To override the group ID of group **kmem** to be **3**, use the following entry.

  ```
  kmem::3:
  ```

- To override the group password and membership for group system to the single account root, use the following entry.

  ```
  system:*::root
  ```

- The following example shows a **passwd_override** file entry that changes a specific machine's password for user **mozart's** account:

  ```
  mozart:sq1Rc1Urrb1L6:678:893:Wolfgang A. Mozart:/aria/wolfgang:/bin/csh
  ```

- Assume that the override file contains the following two entries that override the login shells:

  ```
  mozart::::::/bin/ksh:::25:::/bin/csh
  ```

  If a principal logs in as **mozart,** the override that is keyed by **mozart** is in effect. In this case, principal (**mozart**) is more specific than the group (**25**).

---

passwd_import Operation

Introduction — passwd_import Operation

When **passwd_import** processes entries, it compares group and password file entries to registry entries. It can find two types of conflicts:

- **Name Conflicts** — These conflicts arise when the same name string is defined in the registry and the group and password files. The names **joe 102** and **joe 555** exemplify such a conflict. The duplicate name can represent the same user or two different users.
- **UNIX ID Conflicts** — These conflicts arise when the same UNIX ID is defined in the registry and in the group and password files for users with different names. The names **joe 102** and **ann 102** exemplify such a conflict.

These conflicts can be found separately, as in the preceding examples, or together. For example, a registry entry of **joe 102** and a UNIX entry of **joe 102** are in conflict. When a conflict is found, you must supply either the information that is used to change the password and group files or inform **passwd_import** not to import that entry. The **passwd_import** command makes no changes to existing registry principals, groups, or accounts. However, if you so specify, the **passwd_import** command will create new principals, groups, and accounts in the registry that are based on the group and password files.

More Detailed Information — passwd_import Operation

Registry Entries Created by passwd_import

If an entry exists in the password or group file but does not exist in the registry, **passwd_import** creates a new registry entry. For additional registry information, **passwd_import** takes the following values:

- **For Principal and Group Entries:**
  - `Alias/Primary name` = If the password file contains two entries with the same UNIX number, **passwd_import** creates a primary name entry for the first UNIX number it finds and alias for each occurrence of the same UNIX number
  - `Full Name` = A blank string; no full name is added for the entry
  - `Membership List` = For new groups only, all principals that are listed in the group file and all principals with registry accounts that are associated with that group
  - `Project List` = Yes (for groups only)
- **For Account Entries:**
  - `Account Expiration Date` = None
  - `Account-Valid Flag` = No (use the **dcecp account modify** command to change this flag to **y** after the password is set)
  - `Client Flag` = Yes
  - `Duplicate Ticket Flag` = No
  - `Forwardable Ticket Flag` = Yes
  - `GECOS` = The same value as the entry in the principal's GECOS field in the **etc**/**passwd** file
  - `Good Since Date` = Time of the account creation.
  - `Home Directory` = The same value as the principal's home directory entry in the /**etc**/**passwd** file
  - `Login Shell` = The same value as the principal's login shell entry in the /**etc**/**passwd** file
  - `Maximum Ticket Lifetime` = Set to the registry authentication policy
  - `Maximum Ticket Renewable` = Set to the registry authentication policy.
  - `Password` = Randomly generated (you must modify or reset randomly-generated passwords before user authentication is possible)
  - `Password Date and Time Modified` = Set to the date and time **passwd_import** was run
  - `Password-Valid Flag` = No
  - `Postdated Ticket Flag` = No
  - `Proxiable Ticket Flag` = No
  - `Renewable Ticket Flag` = Yes
  - `Server Flag` = Yes
  - `TGT Authentication Flag` = Yes

Note: **passwd_import** does not set usable passwords for the accounts it creates. You must use the **dcecp account modify** command to set passwords before authentication is possible.

You must also ensure that the account is made valid. You can do this at the same time you set a new password by specifying the **–acctvalid** option on the **dcecp account modify** command, as in the following example:

```
dcecp> account modify John_Hunter -acctvalid {yes}
```

Failure to do so results in the principals' inability to use **dce_login**.

As **passwd_import** processes entries, it performs the following steps in sequence:

1. It opens the group and password files and establishes a connection to the registry.
2. It compares the group file entries to the registry group entries. If there are no conflicts, it creates registry group entries that correspond to the groups.
3. It compares the entries in the password file to the registry principal entries. Again, if there are no conflicts, it:
   a. Creates principals in the registry that correspond to the entries in the password file,
   b. Adds the newly created principals to the appropriate groups, and
   c. Creates accounts for the newly created principals.
4. It re-examines the group file and adds the principals as members of any additional groups that it find there.

The changes to the registry are made individually as each step is processed. If you specify the **–o** option, **passwd_import** add all newly created registry principals to the specified organization. If you do not specify the organization, the principals are added to the organization **none**.

---

passwd_import Usage

Introduction — passwd_import Usage

To use **passwd_import**, the Security server must be running. The following describes **passwd_import** command usage.

More Detailed Information — passwd_import Usage

Using the Identical User Option

The **–i** option lets you specify that duplicate names are not in conflict but, instead, represent the same identity. When **passwd_import** finds duplicate name entries, it processes them as though they are the same user and skips to the next entry.

Using Check Mode

Run **passwd_import** first in check mode by using the **–c** option. In this mode, **passwd_import** attempts to simulate the results of a processing run, showing the conflicts that are encountered when **passwd_import** is run without the **–c** option.

Check mode gives you a good idea of the quantity and complexity of the potential conflicts. However, check mode does not make any changes to the registry. When you run **passwd_import** without the **–c** option and make changes to resolve conflicts, these changes can, in turn, create further conflicts not readily apparent in check mode.

If you encounter numerous conflicts in check mode, it is more efficient to manually edit either the registry or the AIX group and password files to resolve some obvious conflicts before you run **passwd_import**.

Resolving Conflicts

The **passwd_import** command prompts you for instructions on how to resolve the conflicts it finds. You have the following choices:

- You can create an alias to resolve a AIX ID conflict. This action creates an alias for the registry object that is in conflict. This alias is assigned the same name as the conflicting entry in the group or password file. For example, if the entry **joe 555** exists in the registry and the entry **tim 555** exists in the password file, this option creates the alias **tim** for **joe 555**.
- You can generate a new AIX ID automatically or enter a new one explicitly to resolve a AIX ID conflict. For example, if there is a conflict between the entry **joe 555** in the registry and **tim 555** in the password file, you can generate a new AIX ID for **tim**.
- You can enter a new name to resolve a name conflict. For example, if there is a conflict between the entry **joe 555** in the registry and **joe 383** in the password file, you can generate a new name for **joe 383**. This new name will be added to the registry.

In addition, you are given the option of ignoring the conflict and skipping the entry.

Answering Prompts

When you run **passwd_import**, you can be prompted for names and numbers (AIX IDs). Names can contain any characters except the at sign (@) and colon (:) characters, and names must not exceed 1024 characters in length.

If you enter a name or number in an incorrect format, **passwd_import** ignores your entry and prompts you again.

Physical Security of the Database

DCE Security provides safeguards for network security, protecting information that is transmitted across the network by guaranteeing the identities of principals who engage in cross-machine communications. The Security server and the database that it serves, however, reside on a local machine. The registry database is only as secure as the security provided by the machine on which it resides.

In addition to ensuring that sensitive data can be accessed on the local machine only by **root**, you should provide physical security for the machine on which the Security server resides. This can include situating the machine in a locked room, keeping a log of when the server machine is accessed (and by whom) and any other methods that may be appropriate to your needs.

See the *DCE for AIX Application Development Guide — Core Components* for a more detailed discussion of authentication.

Planning Sites for Security Service Components

The first step in configuring the Security Service in your network is to choose the sites for the primary replica and any secondary replicas of the registry. These sites will run **secd** (the security server). Machines running **secd** must be up and available at all times. It is especially important that the machine where the primary replica runs be available throughout the network.

The machine size required to run **secd** depends on the platform and operating system. As a very general rule, choose machines large enough to accommodate future growth of the registry database. The machines must have enough disk space for the registry database and enough backing storage to ensure that processes are not destroyed.

When you run the configuration process, the primary replica site is configured to run the DCE Host daemon (**dced**), which provides the Endpoint Map Service for the local host and any required Cell Directory Service Servers. For information on the configuration process, see *DSS for AIX Up and Running!*.

Policies

Introduction — Policies

You can set policies for:

- The registry as a whole with the **dcecp registry modify** command. The policies thus apply to all principals, groups, and organizations unless a stricter policy is set for specific organizations or accounts.
- Specific organizations with the **dcecp organization modify** command.
- Specific accounts with the **dcecp account modify** command.

There are two types of policies — standard policy and authentication policy.

More Detailed Information — Policies

Standard Policy

Standard policy regulates such things as account and password lifetimes and password format. It can be set for the registry and for specific organizations. The standard policies you can set are described in the following sections.

Note: In addition to defining the password policies described in this section, you can exert additional control in such areas as password formats, password generation, invalid login handling, and expired password handling by attaching ERAs to principals. See “Creating and Maintaining Principals, Groups, and Organizations” on page 1-48 for information on how to do this.

Account Lifespan

The account lifespan that you set determines the period during which the accounts for a specific organization or the registry as a whole are valid. After the period of time passes, the accounts become invalid and must be recreated.

You define the account lifespan as the **dcecp acctlife** attribute in the form:

acctlife {*time***|unlimited}**

where:

time Is a number that indicates the number of days the account is valid.

unlimited Specifies an unlimited lifespan.

An account's lifespan is also controlled by the account expiration date (**expdate** attribute) that you set when you use the **dcecp account create** or **account modify** command to create or change an account. If you set an account expiration date in conflict with the account lifespan policy, the stricter setting applies. For example, if you set the standard policy account lifespan to 40 days and then you set an account expiration date to the next day, the account expires on the next day because that is the stricter setting.

Note: You can control the validity of accounts at a more immediate level by using the **dcecp account modify** command.

Password Lifespan

The password lifespan specifies the period of time before account passwords for a specific organization or the registry as a whole expire.

Generally, DCE Security disables login for users whose passwords have expired. It is possible, however, to override this policy for a user such as *cell_admin*, in order to prevent the cell administrator from being locked out of the system by an expired password. You do this by attaching an instance of the *passwd_override* ERA to the principal. See "Creating and Maintaining Principals, Groups, and Organizations" on page 1-48 for information on how to do this.

You define the password lifespan as the **dcecp pwdlife** attribute in the form:

pwdlife {*time* | **unlimited}**

where:

time Is a number that indicates the number of days the password is valid.

unlimited Specifies an unlimited lifespan.

You can also set the exact date passwords expire by using the password expiration date policy (**pwdexpdate** attribute).

Password Expiration Date

The password expiration date sets the exact date on which account passwords for a specific organization or for the registry as a whole expire.

Generally, DCE Security disables login for users whose passwords have expired. It is possible, however, to override this policy for a user such as *cell_admin*, in order to prevent the cell administrator from being locked out of the system by an expired password. You do this by attaching an instance of the *passwd_override* ERA to the principal. See "Creating and Maintaining Principals, Groups, and Organizations" on page 1-48 for information on how to do this.

You define the password expiration date as the **dcecp pwdexpdate** attribute in the form:

pwdexpdate {*date* | **none}**

where:

date Is the date the password expires in yyyy-mm-dd format.

none Specifies that the password has no expiration date.

You can also set a period of time after which a password expires with the password lifespan policy (**pwdlife** attribute).

Password Format

The password format policies apply to a specific organization or the registry as a whole. They determine the following:

- The minimum length of passwords defined by the **dcecp registry modify pwdminlen** attribute in the form:

  pwdminlin *integer*

  Passwords cannot consist of fewer characters than the number you enter for *integer*. If you specify **0** (zero), no minimum length is in effect.

- Whether or not passwords can consist entirely of spaces defined by the **dcecp pwdspaces** attribute in the form:

  ```
  pwdspaces {yes | no}
  ```

  If you specify **no**, passwords cannot consist of all spaces.

- Whether or not a password can consist entirely of alphanumeric characters defined by the **dcecp pwdalpha** attribute in the form:

  ```
  pwdalpha {yes | no}
  ```

  If you specify **no**, passwords must contain characters other than alphanumerics.

Note: You can exert additional control over password formats by attaching ERAs to principals. For information on how to do this, see "Creating and Maintaining Principals, Groups, and Organizations" on page 1-48.

Populating the New Registry Database

Introduction — Populating the New Registry Database

If the registry database is newly created and started, you must populate the database by setting policies and procedures and adding accounts.

More Detailed Information — Populating the New Registry Database

Setting Policies and Properties

Use the **dcecp registry show** and **dcecp registry modify** commands to view policies and properties and to change them as desired. The **rgy_edit properties**, **rgy_edit policy**, and **rgy_edit authpolicy** commands perform the same functions.

Adding Accounts

After a new registry database is created, it contains only the principals, groups, organizations, and accounts that were added as initial information by **sec_create_db**. Use the **dcecp account create** or **rgy_edit add** command to add any other names and accounts that your site requires. You can do this now or at any time later. See "Creating and Maintaining Accounts" on page 1-48 for information about adding accounts using **dcecp**.

Principal, Group, and Organization Names

Introduction — Principal, Group, and Organization Names

You must assign a name to each principal, group, and organization in the registry. Although a principal, group, and organization can have the same name, no two principals, groups, or organizations can have the same name. For example, two principals cannot be named **smith**, but a principal can be named **smith**, a group can be named **smith**, and an organization can be named **smith**. You can assign up to three types of names — primary names, full names, and aliases.

More Detailed Information — Principal, Group, and Organization Names

Primary Names

Primary names are assigned to principals, groups, and organizations. A registry object's primary name is the name that is used by most system utilities when a human-readable string is needed. When you add a principal, group, or organization to the registry database, you must supply a primary name. The primary name is a key field that you can use as input to the **principal show** command to query the registry database.

Full Names

Full names can be assigned optionally to principals, groups, and organizations. An object's full name is for information purposes. It typically describes or expands a primary name to allow easy recognition by users. For example, a principal could have a primary name of **jsbach** and a full name of **Johann S. Bach**. An organization could have the primary name **moco** and the full name **Motet Composers**.

A full name is a data field only. You cannot use it to query the registry database. You can create the principal's, group's, or organization's full name when you create the principal, group, and organization itself.

Aliases

An alias is an optional alternate name for a primary name. Aliases can be assigned to principals and groups, but not to organizations. Aliases and the primary name they represent share the same UUID and UNIX ID. (UUIDs and UNIX IDs are described in “Universal Unique Identifiers and UNIX IDs” on page 7-145). An alias is a key field that you can use to query the registry database.

Because you can create one account for each primary name and each alias, aliases give you the flexibility of establishing several accounts for the same principal. For example, assume that you create three aliases — **gustav**, **gus**, and **gm** — for the primary name **mahler**. You can then create accounts for the principal **mahler** — one for the primary name and one for each of the name's aliases. The accounts can use different home directories and passwords and can be associated with different groups and organizations.

Since principals accrue only the rights that are associated with the primary name or the alias with which they log in, these multiple accounts for the same person accommodate different access patterns. For example, **mahler** may be a member of the **composers** group and **gustav** can be a member of the **music_admin** group, which is a group of system administrators. The principal **mahler** logs in as **mahler** to perform day-to-day tasks and as **gustav** to perform administrative duties. To help prevent accidental damage to the system, it is a good idea to set up accounts so that users can log into an account with the least privileges necessary to perform their tasks. For groups, aliases are useful if you want to associate two group names with the same UNIX numbers.

See “Universal Unique Identifiers and UNIX IDs” on page 7-145 for information on creating aliases.

Name Formats

Names must not exceed 1024 bytes in length. Names in the registry can contain any characters or digits in the Portable Character Set (PCS), except the at (@) sign and the colon (:) character. The PCS consists of the ASCII characters (encoded by bytes) in the 0x20–0x7E range. This is the base character set and is the default.

Optionally, you can use an extended character set which consists of all characters, including those outside the 0x20–0x7E range, but still excluding the @ and : characters. Names containing extended characters will only be valid when the **DCE_USE_NONPORTABLE_NAMES** environment variable is set to **1**. If this environment variable is not activated, DCE will allow file-handling operations (such as add, delete, replace, and rename) only on names that are restricted to the portable character set. There are two environment variables to enable multibyte characters for CDS: **MB_CUR_MAX** and **DCE_USE_WCHAR_NAMES**.

Note: Names that use characters outside of the portable character set will not be processed correctly if the locale (code set or code page) of a process differs from that of the process under which the name was created. Names using characters outside of the PCS should be used only in environments that are homogeneous and which will remain homogeneous.

Properties

Introduction — Properties

The **dcecp registry modify** command sets properties for the registry as a whole. The properties that you can set are described in "More Detailed Information — Properties."

More Detailed Information — Properties

Default Ticket Lifetime Property

The default ticket lifetime is the default lifetime in hours for tickets that are issued to principals in the registry.

You set default ticket lifetimes with the **dcecp deftktlife** attribute in the form:

deftktlife *DD-hh:mm:ss*:

DD-hh:mm:ss is a number indicating the number of days, hours, minutes, and seconds in the lifetime. If an integer is not entered in this format, the lifetime is set in seconds. For example, **36001** would be 10 hours and 1 second.

Hidden Password Property

The hidden password property determines whether encrypted passwords are displayed or not. You set the hidden password property with the **dcecp hidepwd** attribute in the form:

hidepwd {yes | no}

where:

yes Displays an * (asterisk) in place of the encrypted password in command output and in files where passwords are displayed.

no Displays the hidden password.

Minimum Group ID Property

The minimum group ID property is the starting point for group IDs that are automatically generated by the Security Service when a group's account is added to the registry. (You can explicitly enter a lower group ID than this number; it applies only to automatically generated numbers.)

You set the minimum group ID property with the **dcecp mingid** attribute in the form:

mingid *integer*

integer is the starting ID number.

Minimum UNIX ID Property

The minimum organization ID property is the starting point for organization IDs that are automatically generated by the Security Service when an organization's account is added to the registry. (You can explicitly enter a lower organization ID than this number; it applies only to automatically generated numbers.)

You set the minimum organization ID property with the **dcecp minorgid** attribute in the form:

minorgid *integer*

integer is the starting ID number.

Minimum UNIX ID Property

The minimum UNIX ID property is the starting point for UNIX IDs that are automatically generated by the Security Service when a principal's account is added to the registry. (You can explicitly enter a lower UNIX ID than this number; it applies only to automatically generated numbers.)

You set the minimum organization ID property with the **dcecp minuid** attribute in the form:

minuid *integer*

integer is the starting ID number.

Maximum UNIX ID Property

The maximum UNIX ID property (**maxuid** attribute) lets you set the highest number that can be supplied as a UNIX ID when the accounts for principals are created. (The maximum allowable UID is 65535.) This maximum applies to both the system-generated and user-entered UNIX IDs.

You set the maximum UNIX ID property with the **dcecp maxuid** attribute in the form:

maxuid *integer*

integer is the starting UNIX ID.

Minimum Ticket Lifetime Property

The minimum ticket lifetime is the minimum amount of time in minutes before the principal's ticket must be renewed. This renewal is performed automatically with no intervention on the part of the user. The shorter you make the minimum ticket lifetime, the greater the security of the system. However, extremely frequent renewal can degrade system performance. The minimum ticket lifetime that you set needs to take into consideration system performance and the level of security that your cell requires.

You set the minimum ticket lifetime with the **dcecp mintktlife** attribute in the form:

mintktlife *integer*

integer is a number that indicates the number of seconds in the minimum ticket lifetime.

The minimum ticket lifetime can be set only as a registry property. It cannot be set for individual accounts. (Contrast this with the maximum ticket lifetime property, which is set with the **dcecp registry modify** or **account modify** commands.)

Displaying and Setting Properties

To display registry properties, use the **dcecp registry show** command.

Example — Properties

To set registry properties, use the **dcecp registry modify** command. The following sample command uses the **maxuid** option to change the maximum UNIX ID property to 67899:

```
dcecp> registry modify -maxuid 67899
```

Note that the example shown above uses an attribute option. You can also set properties by using the **dcecp registry modify** command with the **–change** option and attribute lists. For example, to use an attribute list to set the maximum UNIX ID property to 67899, the command would be:

```
dcecp> registry modify -change {maxuid 67899}
```

Recovering Slave Replicas

Introduction — Recovering Slave Replicas

Because slave replicas are not backed up, you must recreate a replica to restore a replica that is corrupted. To recreate a replica, use the steps in this section.

Steps — Recovering Slave Replicas

1. Use standard AIX commands to manually delete the replica's database files and master key file. To do this, delete all the files in /**opt**/**dcelocal**/**var**/**security**/**rgy_data** and the file in /**opt**/**dcelocal**/**var**/**security**/**.mkey**.
2. Use the **set _s(sec)** command to bind to the master and then the **dcecp registry delete –force** command to delete the replica from the master's replica list. The following commands first bind to the master /**.../musee.com**/**subsys**/**dce**/**sec**/**master** and then delete the replica /**.../musee.com**/**subsys**/**dce**/**sec**/**art**.

   ```
   dcecp> set _s(sec) /.../musee.com/subsys/dce/sec/master
   dcecp> registry delete /.../musee.com/subsys/dce/sec/art -force
   dcecp>
   ```

3. Use standard AIX commands to copy the file /**opt**/**dcelocal**/**etc**/**security**/**pe_site** from the machine running the master to the machine that will run the replica.
4. Use the configuration process on the replica machine to:
   a. Stop DCE daemons
   b. Start DCE daemons
   c. Configure a Security server replica. This configuration creates the replica's database and starts **secd**. See *DSS for AIX Up and Running!* for detailed configuration information.
5. When you configure the replica in the previous step, you assign it a name. If you did not give this replica the same name that it previously had, the old name must be deleted from CDS by performing the following steps:
   - Deleting the replica's server entry name from /**.:**/**subsys**/**dce**/**sec**

- Deleting the replica's name from the CDS group /**.:/sec**

Recovering the Master Replica

Introduction — Recovering the Master Replica

To recover a master replica because the master replica's database is damaged, you can use either of the following methods:

- Use the **dcecp registry designate master** command to make a slave replica the master replica and create a slave replica on the host of the former master replica. This method is described in a section below.
- Restore the master replica from a backup. This method is described in “Performing Routine Maintenance” on page 1-58.

The method you choose depends on whether the master replica's backup database or the slave replica's database is more current.

More Detailed Information — Recovering the Master Replica

Determining the Most Current Database

To determine whether the backup of the master replica's database or a slave replica's database is more current, run the **dcecp registry show –replica** for the replica. The output of this command lists the last update sequence number, date, and time. Compare the replica's last update sequence number, date, and time with the sequence number, date, and time of the master replica's backup. If the replica is more current, make the replica the new master as described in the following section. If the master replica's backup is more current, restore the master from the backup as described in “Performing Routine Maintenance” on page 1-58.

Turning a Slave Replica into the Master Replica

This section describes how to use the **dcecp registry designate –master** command to turn a slave into the master. Be aware that because the **registry designate –master** command can cause data to be lost, the **registry designate** command without the **–master** option is the preferred means of designating a different master replica. Use the **registry designate –master** command only if the master replica is irrevocably damaged and you are unable to use **registry designate** command without the **–master** option.

Steps — Recovering the Master Replica

Follow these steps to turn a master replica into a slave replica:

1. Choose the slave replica that will become the new master replica.
2. If necessary, issue the **set _s(sec)** command to set the chosen slave to the default replica. The following example sets the default replica to /**.../musee.com/subsys/dce/sec/art**:

   ```
   dcecp> set _s(sec) /.../musee.com/subsys/dce/sec/art
   dcecp>
   ```

3. Issue the following **registry designate –master** command to change the default host to the master registry:

   ```
   dcecp> registry designate -master
   dcecp>
   ```

4. Use the **registry show –replica** command to verify the change.
5. Use standard AIX commands to delete the old master replica's database and **.mkey** file by deleting the /**opt**/**dcelocal**/**var**/**security**/**rgy_data** directory and the /**opt**/**dcelocal**/**var**/**.mkey** file.
6. Use the **registry delete** command with the **–force** option to remove the old master from the replica list. The following example deletes the old master named **history** from the replica list:

```
dcecp> registry  delete -force /.../musee.com/subsys/dce/sec/history
dcecp>
```

Recovering the Registry

If the registry becomes inoperable or unavailable for an extended time, you may need to restore it. See “Restoring the Registry” under “Steps — Backing Up and Restoring the Registry Database” on page 7-36 for more information.

Registry ACL Manager

Introduction — Registry ACL Manager

The registry ACL Manager consists of nine manager types. These manager types handle different ACL semantics that are required by the five types of objects in the registry. For example, the Principal ACL Manager type controls the ACLs on all **principal** objects in the registry. Because **group** objects require a set of permissions that are different than those of a **principal** object, there is a separate Group ACL Manager type that controls the ACLs on **group** objects.

Permissions Required — Registry ACL Manager

Not all permissions, nor all ACL entry types, are valid for each ACL Manager. Table 7-6 on page 7-113 summarizes the valid and invalid permissions and the invalid ACL entry types for each ACL Manager.

Table 7-6. ACL Managers, Valid Permissions, and ACL Entry Types		
Manager Type	**Valid Permissions**	**Invalid ACL Entry Types**
dir (controls directory objects)	rcidDn	user_obj group_obj
policy (controls the policy object, not supported in this release)	rcma	user_obj group_obj
principal (controls principal objects)	rcDnfmaug	group_obj
group (controls group objects)	rctDnfmM	user_obj
org (controls org objects)	rctDnfmM	user_obj group_obj
replist (controls the replist object, not supported in this release)	cidmAI	user_obj group_obj
secdirectory (controls registry directory objects, not supported in this release)	rcidDn	user_obj group_obj
srvrconf (controls server configuration objects, not supported in this release)	rcdfwx	user_obj group_obj
xattrschema (controls extended registry attribute types, not supported in this release)	rcidm	user_obj group_obj

Registry Editor Usage

Introduction — Registry Editor Usage

Although you can use the DCE control program to maintain the network-wide Registry, you can only use the Registry Editor to maintain the local registries located on the hosts in a cell.

The following sections explain how to start, stop, and get help for the Registry Editor and describe the commands to use for local registry maintenance. Specific instructions for using the Registry Editor to maintain the local registry are given in “Creating and Maintaining Accounts” on page 1-48.

For detailed descriptions of all of the Registry Editor commands, see the **rgy_edit** pages in the *DCE for AIX Administration Command Reference*.

More Detailed Information — Registry Editor Usage

Starting, Stopping, and Getting Help

The Registry Editor runs in two modes — interactive and command line. In interactive mode, the control program prompts you for the information that it needs. In command-line mode, you enter all of the information that the control program needs on the command line. In command-line mode, you can perform only one operation at a time; however, you may find command-line mode useful for creating shell scripts that execute a sequence of Registry Editor commands. Most of the examples in this guide are in interactive mode. (See the **rgy_edit** reference page in the *DCE for AIX Administration Command Reference* to invoke and use the Registry Editor in command-line mode.)

The **rgy_edit help** command displays help information. If you enter `help` or `h`, the Registry Editor displays a list of all commands and available topics. For example:

```
rgy_edit=> help
```

rgy_edit Commands for Local Registry Maintenance

To view or to perform any maintenance tasks on the local registry, you must first access it. To access the local registry, invoke the Registry Editor with the **–l** option:

```
$ rgy_edit -l
rgy_edit=>
```

At the `rgy_edit` prompt, enter the name of the command for the particular operation you want to perform. The following list describes the available commands.

Command	Function
del[ete]	Deletes the entries for principals, groups, or organizations.
prop[erties]	Displays or changes local registry properties.
p[urge]	Deletes expired entries for principals, groups, or organizations.
v[iew]	Displays the entries for principals, groups, or organizations.

For detailed descriptions of all of the Registry Editor commands, see the **rgy_edt** pages in the *DCE for AIX Administration Command Reference*.

Steps — Registry Editor Usage

- To start the Registry Editor in interactive mode, enter the following command:

  ```
  $ /opt/dcelocal/bin/rgy_edit
  ```

  The Registry Editor responds by displaying the name of the current registry site and the `rgy_edit=>` prompt, as follows:

  ```
  Current site is:
  registry server at /.../bayre.com/subsys/dce/sec/master
  rgy_edit=>
  ```

- To exit from a **rgy_edit** command, press <Enter> at the command prompt. For example, to exit from the **add** command to add principals, press <Enter> at the `Add Principal=> Enter name:` prompt.
- To exit from the Registry Editor, enter the **q[uit]** command at the `rgy_edit` prompt:

  ```
  rgy_edit=> q
  $
  ```

Example — Registry Editor Usage

If the name service is unable to provide the name, the output is shortened. For example, instead of `registry server at /.../bayre.com/subsys/dce/sec/master`, the display would be `registry server at /.../bayre.com`.

Registry Permissions

Introduction — Registry Permissions

The following list describes the permissions that can be granted the object types found in the registry.

Permission	Meaning
A	Execute commands that act on replicas (**sec_admin**).
a	Modifies authentication information.

c	Modifies ACLs on objects. All registry ACLs must have one entry that specifies **c** (control) permission.
d	Deletes from an object's contents.
D	Deletes an object from the registry.
f	Modifies a principal's, group's, or organization's full name.
g	Adds a principal to a group.
i	Adds to an object's contents.
m	Modifies management information.
M	Adds and deletes members from this group or organization. To add a member to a group, you must also have **g** permission for the principal to be added.
n	Modifies the name of a directory, a principal, a group, or an organization.
r	Views management, authentication, and user information.
t	Tests a group or organization membership of a named principal.
u	Modifies user information.

More Detailed Information — Registry Permissions

Management, Authentication, and User Information

The registry contains three different kinds of information about the objects in it — management information, authentication information, and user information. The specific pieces of information that are kept for each object type are summarized in the following sections.

Management Information

Management information includes the following categories:

- **For Registry Policies and Properties**
  - The account life span
  - The password minimum length
  - The password lifespan
  - Whether passwords can contain spaces
  - Whether passwords can consist of nonalphanumeric characters
  - The password expiration date
  - The minimum ticket lifetime
  - The default ticket lifetime
  - A number that defines the lowest UNIX ID that is supplied automatically when principals, groups, or organizations are created
  - A number that defines the highest number that can be supplied (either automatically or manually) as a UNIX ID when principals, groups, or organizations are created
  - Whether encrypted passwords are displayed (the shadow password property)
- **For Principals**
  - Object creation quota for the principal

  - Whether the principal can change primary names to aliases and aliases to primary names
  - The account, group, and organization names
  - The expiration date for the principal's account
  - The Account-Valid flag for the principal's account
  - Flags that indicate whether the account is for a principal that can act as a server or as a client
  - Text string showing the full name of the principal
  - User Identifier (UID) of the principal
  - Unique User Identifier (UUID) of the principal
- **For Groups**
  - Whether the group's primary name can be changed to an alias and its aliases to its primary name
  - The project list inclusion property
  - Primary name of the group
  - Text string showing the full name of the group
  - Group Identifier (GID) of the group
  - Unique User Identifier (UUID) of the group
- **For Organizations**
  - Whether the organization's primary name can be changed to an alias and its aliases to its primary name
  - The account lifespan
  - The password minimum length
  - The password lifespan
  - The password expiration date
  - Whether passwords can contain spaces
  - Whether passwords can consist of nonalphanumeric characters
  - Primary name of the organization
  - Text string showing the full name of the organization
  - Organization Identifier (ORGID) of the organization
  - Unique User Identifier (UUID) of the organization
- **For the xattrschema object**

  Whether or not the **xattrschema** can be modified.

Authentication Information

Authentication information includes the following categories:

- **For Registry Policies and Properties**
  - The maximum ticket lifetime
  - The maximum time for which tickets can be renewed
- **For Principals**
  - The maximum ticket lifetime for the principal's account

- The maximum time for which tickets issued to the principal's account can be renewed
- The date and time that the principal's account was last changed (Good Since Date)
- Whether the principal's account can be issued postdated tickets, forwardable tickets, renewable tickets, or proxiable tickets
- Whether the Authentication Service can issue tickets to the principal's account based on ticket-granting ticket authorization or whether principals must obtain tickets directly for the service
- Whether the principal's account can be issued duplicate session keys
- The date and time that the principal's account was enabled (Last Changed Date)
- The creator of the principal's account and account creation date
- Description of the account's use

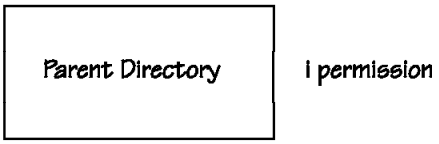
User Information

User information includes the following information pertaining to a principal's account:

- Password
- Home Directory
- Miscellaneous Information (GECOS Information)
- Login Shell
- Password-Valid Flag

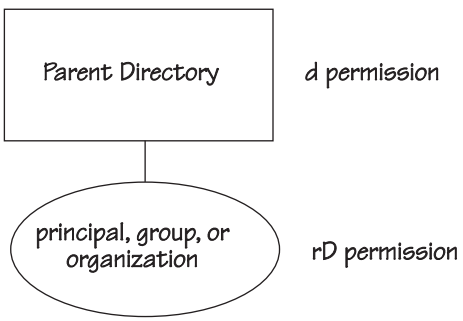
Permissions Required — Registry Permissions

Permission Required to Create Principals, Groups, or Organizations



To create a principal, group, or organization, you must have the **i** permission on the directory in which you create the principal, group, or organization.

Permissions Required to Delete Principals, Groups, or Organizations



To delete principals, groups, or organizations, you must have the following permissions:

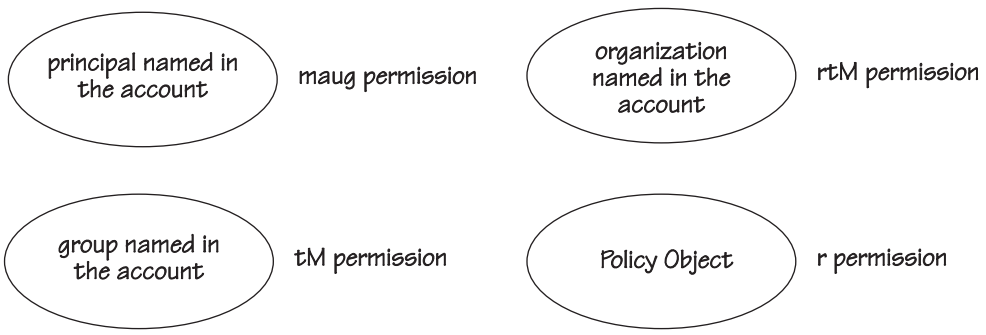
- The **d** permission on the directory in which the principal to be deleted exists.
- The **rD** permission on the principal, group, or organization to be deleted.

Permissions Required to Add Accounts

When you add accounts, the **dcecp** or **rgy_edit** command adds the principal to the group or organization that is named in the account (if the principal is not already a member of the group and/or an organization). For this reason, the permissions that are required to add an account may include the permissions that are required to add a member to a group or organization. The following topics are covered in the discussion of the permissions required to add accounts:

- The permissions required to add an account and, at that same time, add the principal as a member of the group and organization named in the account (see “Adding an Account and the Account Principal to the Group and Organization” in this section).
- The permissions required to add an account for which the principal is already a member of the named group and organization (see “Adding an Account for Which the Principal Is Already a Member of the Group and Organization” in this section).
- The permissions required to add an account and add the principal only to the group that is named in the account (because the principal is already a member of the organization) (see “Adding an Account and the Principal to the Group Only” in this section).
- The permission required to add an account and add the principal only to the organization named in the account (because the principal is already a member of the group) (see “Adding an Account and the Principal to the Organization Only” in this section).

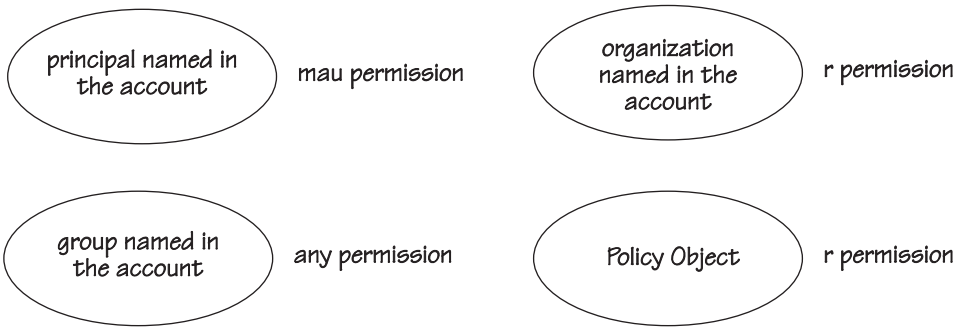
Adding an Account and the Account Principal to the Group and Organization



To add an account and add the account's principal to the group and the organization named in the account automatically, you must have the following permissions:

- The **maug** permissions on the account's principal
- The **tM** permissions on the group that is named in the account
- The **rtM** permissions on the organization that is named in the account
- The **r** permission on the registry **policy** object

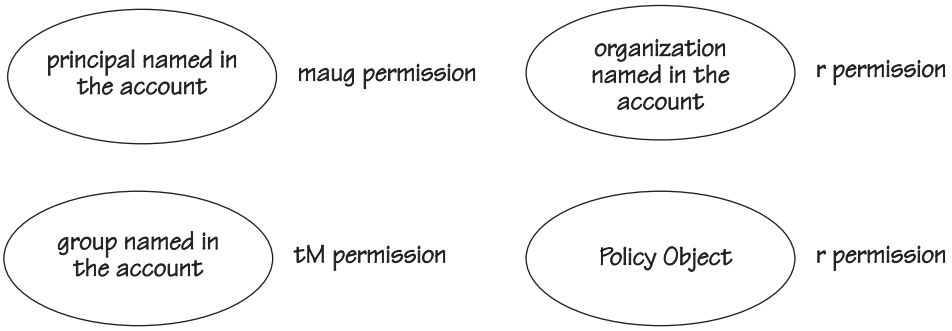
Adding an Account for Which the Principal Is Already a Member of the Group and Organization



To add an account that does not require adding the account's principal to the group and the organization named in the account, you must have the following permissions:

- The **mau** permissions on the account principal
- At least one permission of any kind on the group that is named in the account
- The **r** permission on the organization that is named in the account
- The **r** permission on the registry **policy** object

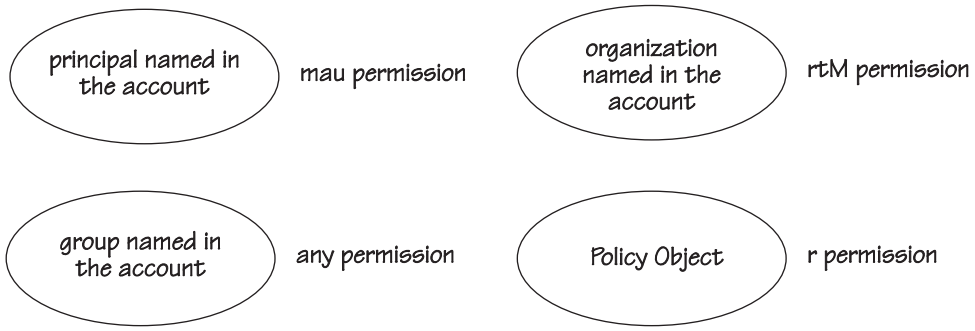
Adding an Account and the Principal to the Group Only



To add an account and add the account's principal to the group (the principal is already a member of the organization named in the account), you must have the following permissions:

- The **maug** permissions on the account's principal
- The **tM** permissions on the group that is named in the account
- The **r** permission on the organization that is named in the account
- The **r** permission on the registry **policy** object

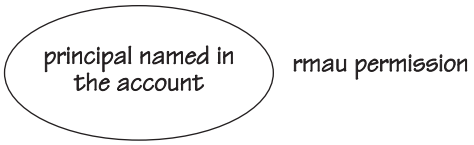
Adding an Account and the Principal to the Organization Only



To add an account and add the accounts principal to the organization (the principal is already a member of the group named in the account), you must have the following permissions:

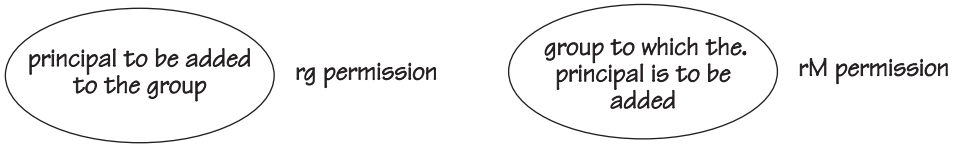
- The **mau** permissions on the account's principal
- At least one permission of any type on the group that is named in the account
- The **rtM** permissions on the organization named in the account
- The **r** permission on the registry **policy** object

Permissions Required to Delete Accounts



To delete accounts, you must have the **rmau** permissions for the principal that is named in the account. you must have the **rmau** permissions for the **preludes**/**villa**/**lobos**.

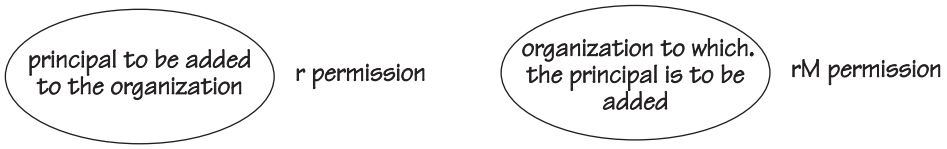
Permissions Required to Add Members to Groups



To add members to groups, you must have the following permissions:

- The **rM** permissions on the group to which the principal is being added
- The **rg** permissions on the principal to be added

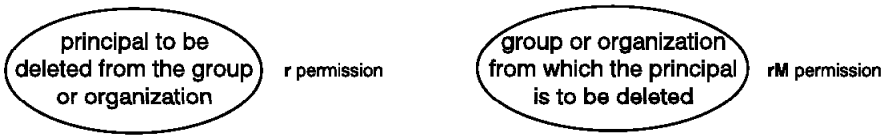
Permissions Required to Add Members to Organizations



To add members to organizations, you must have the following permissions:

- The **rM** permissions on the organization to which the principal is being added
- The **r** permission on the principal to be added

Permissions Required to Delete Members from Groups or Organizations

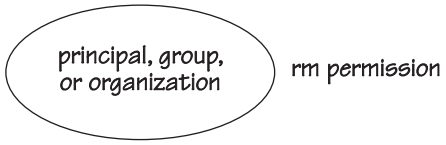


To delete members from a group or organization, you need the **rM** permissions on the group or organization from which the principal is being deleted and the **r** permission on the principal which is being deleted.

Permissions Required to Change a Principal's, Group's, or Organization's Full Name

To change a principal's, group's, or organization's full name, you must have the **rf** permissions for the principal, group, or organization for which you are making the change.

Permissions Required to Change Management Information for Principals, Groups, or Organizations



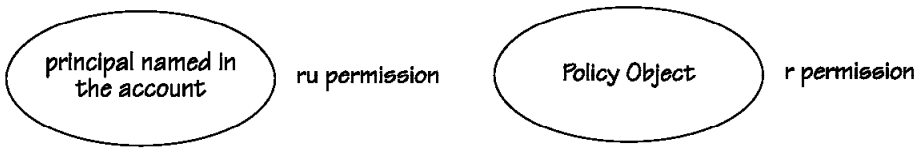
To change management information for a principal, a group, or an organization, you must have the **rm** permissions for the object for which you are changing management information.

Permissions Required to Change Management, Authentication, and User Information (Except Passwords) for Accounts

To change management, authentication, and user information (except passwords) for accounts, you must have the following permissions for the principal named in the account:

- The **ra** permissions to change authentication information
- The **rm** permissions to change management information
- The **ru** permissions to change user information

Permissions Required to Change Passwords for Accounts



To change passwords for accounts, you must have the following permissions for the principal that is named in the account:

- The **ru** permissions on the account's principal
- The **r** permission on the registry **policy** object

Permissions Required to Change Authentication and Management Information for Registry Policies and Properties

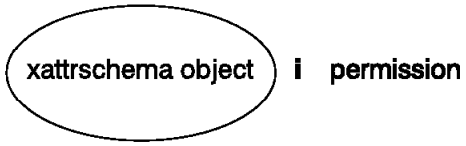
To change management or authentication information for the registry by using the **dcecp registry modify**, **po**, and **auth** commands, you must have the **ra** permissions to change authentication information or the **rm** permissions to change management information permission for the registry **policy** object.

Permissions Required to Execute Commands That Act on Replicas

To execute any of the commands that act on replicas, you must have the following permissions on the **replist** object:

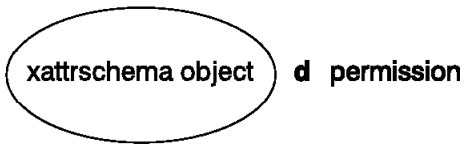
- The **A** permission to execute all commands except for those that display replica information, which require no permissions on the **replist** object
- The **d** permission to execute the commands that delete replicas

Permissions Required to Create Extended Registry Attribute Types



To create an extended registry attribute type in the registry schema, you must have **i** permission on the **xattrschema** object.

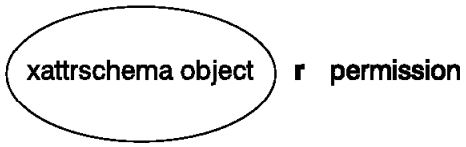
Permissions Required to Delete Extended Attribute Types



To delete extended attribute types, you must have **d** permission on the **xattrschema** object.

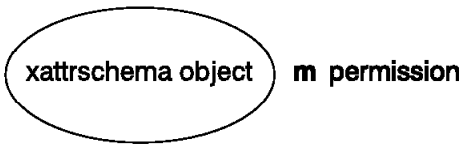
Permissions Required to View Extended Registry Attribute Types

The figure below shows the permission that is required to view one or more ERAs in the registry's schema database (with the **dcecp schema show** command).



To view extended registry attribute types, you must have **r** permission on the **xattrschema** object.

Permissions Required to Modify Extended Registry Attribute Types



To modify extended registry attribute types, you must have **m** permission on the **xattrschema** object.

Permissions Required to Change ACLs on Registry Objects

To modify ACLs on registry objects, you must have the **c** permission on the object whose ACL you are changing. The registry object can be the **policy** object or a principal, group, or organization.

Permissions Required by Slave Replicas

In order to initialize and function properly, slave replicas must have the **i**, **m**, and **l** permissions for the **replist** object (**/.:/sec/replist**). A secondary server runs under the identity of the machine on which it runs. A machine name is the local host principal name in the following form:

host/*hostname***/self**

The required ACL entry is added when the configuration tool initially configures the DCE cell's Security server and when you use the tool to create new slave replicas. The entry has the following form:

user:host/*hostname***/self:imI**

Examples — Registry Permissions

- To create the principal **preludes/villa/lobos**, you must have **i** on **villa**.
- To create an account for the principal **preludes/villa/lobos** associated with the group **composers** and organization **pianist**, you must have the following permissions:
  - The **maug** permissions on **preludes/villa/lobos**
  - The **tM** permissions on the group **composers**
  - The **rtm** permissions on the organization **pianists**
  - The **r** permission on the registry **policy** object
- To create an account for the principal **preludes/villa/lobos** associated with the group **composers** and the organization **pianists**, you must have the following permissions:
  - The **mau** permissions on **preludes/villa/lobos**
  - At least one permission of any kind on the group **composers**
  - The **r** permission on the organization **pianists**
  - The **r** permission on the registry **policy** object
- To delete the principal **preludes/villa/lobos**, you must have the **d** permission for the **preludes/villa** directory, and **rD** permission for the principal **preludes/villa/lobos**.
- To add the principal **preludes/villa/lobos** to the group **composers**, you must have the following permissions:

  - The **rM** permissions on the group **composers**
  - The **rg** permissions on the principal **lobos**
- To add the principal **preludes/villa/lobos** to the organization **pianist**, you must have:
  - The **rM** permissions on the organization **pianists**
  - The **r** permission on the principal **lobos**
- To delete the principal **preludes/villa/lobos** from the group **composers**, you must have the following permissions:
  - The **rM** permissions on the group **composers**
  - The **r** permission on the principal **lobos**

Removing a Server Machine from the Network

Introduction — Removing a Server Machine from the Network

If you are planning to remove a machine that runs a secondary replica from the network or to shut the machine down for an extended period, delete the replica at that site.

If you are removing a node running the primary server, you must change the primary server site as described previously before you remove the node.

Use the **rmdce** command to delete a secondary replica. You must be on the host in order to execute this command that contains the replica that is to be deleted:

```
rmdce -o full sec_srv
```

When this command runs, the primary performs the following actions:

1. Marks the replica as deleted
2. Propagates the deletion to all replicas on its replica list
3. Delivers the delete request to the replica
4. Removes the replica from its replica list
5. The configuration information for the replica is cleaned up on the replica's host.

Verify that the secondary is deleted by issuing the **dcecp registry catalog** command. When the primary replica has received the request to delete the secondary, the secondary displays on the replica list as marked for deletion. When the replica has actually been deleted, the list no longer displays that replica.

Note: Even if you use the **dcecp registry delete** command, you must also unconfigure the replica on the local host with the **rmdce** command.

Example — Removing a Server Machine from the Network

The following sample command deletes the secondary replica named **/.../giverny.com/subsys/dce/sec/art_1**

```
dcecp> registry delete /.../giverny.com/subsys/dce/sec/art_1
dcecp>
```

When you issue this command, **dcecp** binds to the primary replica that is in the current cell, if necessary; then the primary replica instructs the secondary replica to delete itself.

Replicated Databases

Introduction — Replicated Databases

The registry database can be replicated within its cell. Each instance of a Security server in a cell maintains a working copy of the database. Throughout this guide, the combination of a Security server and its data (the registry database) is referred to as a *replica*. You can create several replicas in a cell to enhance performance and reliability.

The task of keeping the cell's replicas consistent is handled automatically by the Security servers. Any changes that you make are automatically reflected in all replicas.

More Detailed Information — Replicated Databases

How Updates Are Handled

Updates are made to only one database (the master) and the changes are propagated to all others (the slaves). While propagations are pending, all replicas are accessible even if they are not completely up to date. In other words, even replicas to which the changes were not yet applied are available. This replication mechanism ensures that all replicas remain available for login validation and for read operations even when changes are in the process of being propagated.

Master and Slave Replicas

Only one replica in a cell (the *master replica*) accepts updates to its database from clients. Other replicas (called *slave replicas*) accept only reads from clients. The master replica propagates any updates to the slave replicas. For example, either a master or a slave replica can provide account information to a client program such as /**bin/login**. However, if you are adding an account or changing password information, those updates can be handled only by the master replica.

The process of updating the database differs slightly between the master replica and the slave replicas. Figure 7-6 on page 7-127 and Figure 7-7 on page 7-128 illustrate the master and slave update processes. The processes are described in the sections that follow the figures.

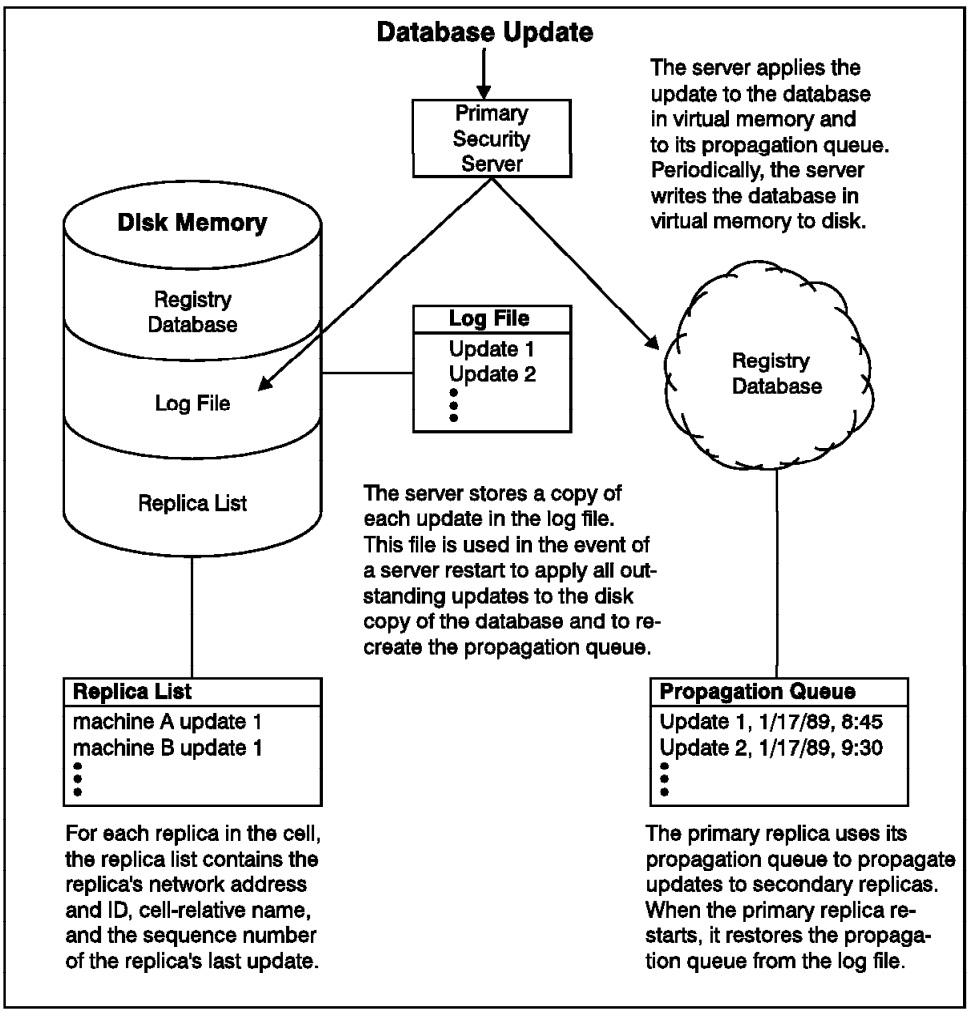


Figure 7-6. The Master Replica Update Process

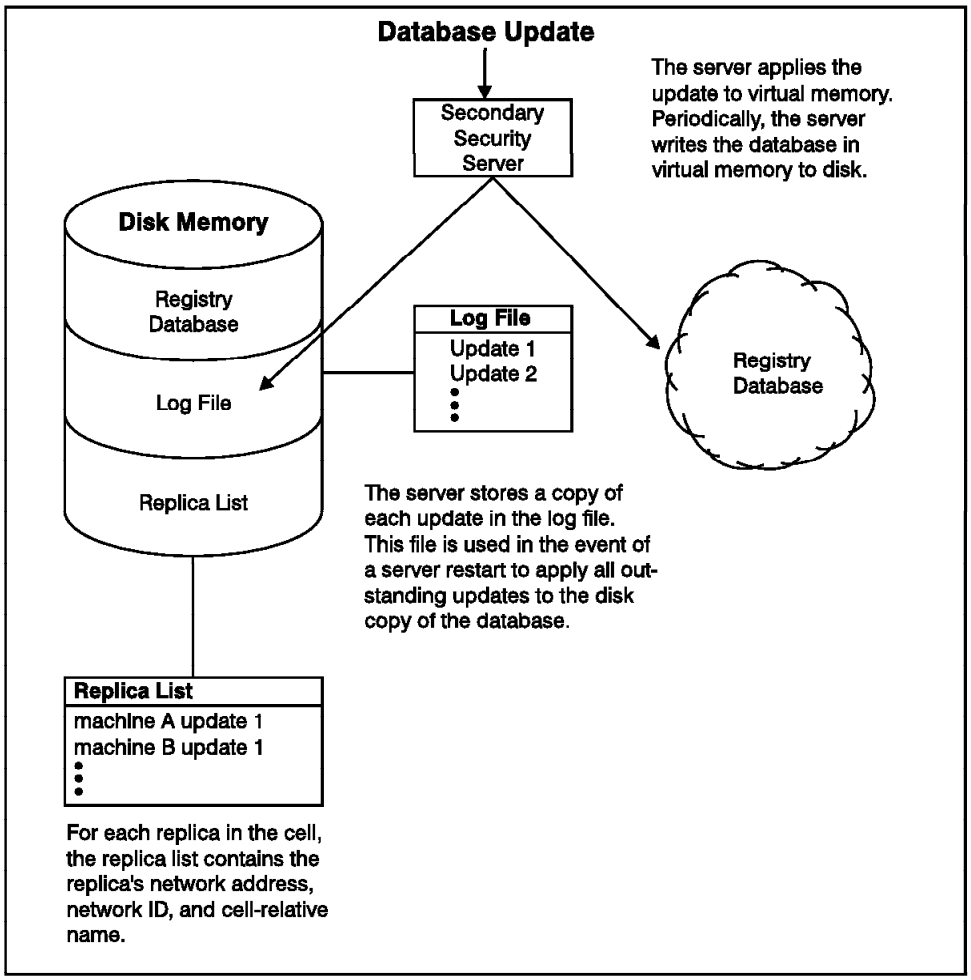


Figure 7-7. The Slave Replica Update Process

Handling Database Updates

When a master or slave replica receives updates, it applies the updates to its database in virtual memory. It also saves a copy of each update in a log file that is stored on the disk. Updates accumulate in the log file in sequence number order. If a server restarts unexpectedly, the log file ensures that no updates are lost.

The replica periodically writes the database in virtual memory to disk, thus bringing the disk copy up to date. Then, if the replica is a slave, it clears the log file of all updates. If the replica is the master, it clears the log file of all updates that have been propagated to the slave replicas. Updates that have not been propagated to the slaves are retained and used to reconstruct the propagation queue, if necessary.

Only the master replica maintains a propagation queue, which is used to hold changes to be propagated to the slave replicas, as described in “Propagating Database Changes” in this section. When the master replica receives an update, it adds it to the propagation queue in addition to its virtual memory database and its log file. Each update in a propagation queue is identified by a sequence number and a timestamp. The sequence number is used internally to track the propagation of updates to slave replicas. The timestamp is provided to show users the date and time of updates.

When a master or slave replica restarts, it initializes its database in virtual memory and then applies any outstanding updates in the log file to its database. If the replica is the master replica, it also recreates the

propagation queue from the log file so that any outstanding slave updates can be propagated. This mechanism ensures that no updates are lost when a server is shut down.

Propagating Database Changes

To propagate updates to the slave replicas, the master replica first updates its copy of the database by using the process described in “Handling Database Updates” in this section. The master replica then attempts to propagate the update to each slave replica on its replica list. The replica list contains each slave replica's ID and network address. It also contains the sequence number of the last update made to the slave.

The master replica always propagates in sequence number order. By examining the sequence number that is associated with a replica in its replica list, and the sequence numbers of the updates that are in its propagation queue, the master can determine which of the updates on its propagation queue must be propagated to which slave. This mechanism helps ensure that the unavailability of a single slave replica does not interfere with updates to the rest of the slave replicas.

If the propagation of an update does not succeed on the first attempt, the master replica tries periodically until it succeeds. This process should take approximately 30 seconds. Therefore, be sure to allow the necessary time before starting lookups of new information. When the update succeeds, the master updates the sequence number that is associated with the updated replica on its replica list. When an update is propagated to all the slave replicas, the master removes the update from its propagation queue.

Master and Slave Authentication

Like all DCE objects, the master and slave replicas must authenticate to each other. To do this, the master carries the identity of **dce_rgy**, one of the principals that is created when the database is created. Slaves carry the identity of the host machine on which they exist. This identity must have the rights to the **/.:/sec/replist** object described in “Registry Permissions” on page 7-114.

Reserved Principals and Accounts

Introduction — Reserved Principals and Accounts

Some principals and accounts are reserved for use by various system operations. You cannot delete reserved principals. You can modify, but not directly delete, reserved accounts. Note, however, that you may delete reserved accounts indirectly by deleting the group or organization that is specified in the account. See “Deleting Accounts” under “More Detailed Information — Adding Accounts” on page 7-10 for details.

More Detailed Information — Reserved Principals and Accounts

A list of reserved principals and accounts follows. In the list *cell_name* is the name of your cell and *host_principal_name* is the name of the machine principal. The actual form of this name is set during DCE configuration.

- **Reserved Principals:**
  - **dce-ptgt**
  - **krbtgt**/*cell_name*
  - **dce-rgy**
  - *host_principal_name*
- **Reserved Accounts:**

- **dce-ptgt none none**
- **krbtgt**/*cell_name* **none none**
- **dce-rgy none none**
- *host_principal_name* **none none**

Restarting a Security Server in Locksmith Mode

Introduction — Restarting a Security Server

To restart a Security server in locksmith mode, perform the steps in this section on the node on which the primary replica is running. You must have **root** access to this node.

More Detailed Information — Restarting a Security Server

The Security server normally runs in the background. When you start a Security server in locksmith mode, it runs in the foreground so that you can answer prompts.

After the Security server is started in locksmith mode, you can use the **dcecp registry modify** command to change the registry so that the standard privileged account can access it. After these changes are made:

1. Shut down the Security server that is running in locksmith mode.
2. Restart a Security server according to your standard procedures.

Steps — Restarting a Security Server

1. Shut down the Security server.
   a. If you cannot log in with administrative privileges and access the **dcecp** command to shut down the server, log in as **root** on the machine on which the server is running and kill the Security server process.
   b. If you are able to log in with administrative privileges, use the **dcecp registry stop** command to shut down the Security server. When you use this command, you must supply the fully qualified name of the replica to stop as an argument. The following sample command stops the replica named **slave_3**.

      ```
      dcecp> registry stop  /.../giverny.com/subsys/dce/sec/slave_3
      ```

2. Start the Security server in locksmith mode. The following example shows the Security server started with the locksmith account that was created for the principal named **master_admin**. The **–remote** option is also supplied to allow **master_admin** to log in from a remote node; otherwise, **master_admin** must log in from the node on which the Security server was started.

   ```
   $ /opt/dcelocal/bin/secd -locksmith master_admin -remote
   ```

   If the locksmith account exists but you have lost its password, use the **–lockpw** option to cause **secd** to prompt you for a new locksmith password and replace the existing password with the one you enter.

Restarting Security Servers

To restart a Security server (primary or secondary), enter the following command:

```
$ /opt/dcelocal/bin/secd &
```

For convenience, set up the server to start automatically whenever the machine reboots.

Restarting the Primary Server in Locksmith Mode

The **secd –locksmith** option starts **secd** in locksmith mode. This option can be used only on the primary replica. In locksmith mode, the principal name that you specify to **secd** becomes the locksmith principal. As the locksmith principal, you can repair malicious or accidental changes that prevent you from logging in with full registry access privileges.

When you bring up a Security server in locksmith mode, **secd** automatically creates a locksmith account or, if the locksmith account exists, it lets you supply a new password for that account. When the Security server is running, you can log into the locksmith account by using the newly changed password, if you changed it. You can then access the registry to change the account or policy information that may have prevented you from accessing the registry using your normal credentials.

In locksmith mode, all principals with valid accounts can log in and operate on the registry with normal access checking. The locksmith principal, however, is granted special access to the registry; no access checking is performed for the authenticated locksmith principal. This means that, as the locksmith principal, you can operate on the registry with full access.

Restoring a Duplicate Master

Introduction — Restoring a Duplicate Master

This section describes how to recover from a very unusual problem. Do not use the methods described here to resolve the problem unless it is absolutely necessary.

More Detailed Information — Restoring a Duplicate Master

Occasionally the replica that you want to be the master has a master sequence number that is lower than (or equal to) another master sequence number in the system. When the master detects that its master sequence number is lower than another one in the system, it marks itself as a duplicate master and its process exits. Each time you start the master replica, it notices that it has been deemed a duplicate master, and its process again exits.

To force this duplicate master to become the master and not exit, restart its **secd** with the **–master_seqno** option in the following format:

secd -master_seqno *new_master_seqno*

new_master_seqno is a new master sequence number to assign to the replica. Make this number one digit higher than the highest master sequence number in the system.

Use the **dcecp registry show –replica** command for each replica to find the highest master sequence number.

Sample passwd_import Session

Introduction — Sample passwd_import Session

This section shows a simplified **passwd_import** session. The sample session uses the following registry group and password entries and the UNIX group fit and password file entries. For convenience, the registry entries are shown in the password and group file format, although they are not stored that way in the registry database.

More Detailed Information — Sample passwd_import Session

Invoking passwd_import

In the sample session, the following **passwd_import** command is entered at the shell prompt:

```
$ passwd_import -d sys5.3_tapes/adm -i -v -u cell admin
```

This command specifies that:

- Identical names represent the same identity (**–i**).
- The UNIX group and password files are in the **sys5.3_tapes/adm** directory.
- The command will not run in check mode (**–c** is not specified).
- The command will run in verbose mode (**–v**).
- The principal whose account should be used for authentication is **cell_admin**.
- The command prompts you for the **cell_admin** account's password because the **–p** option is not used.

After the command is invoked, the system prepares for **passwd_import** processing by displaying the following:

```
Preparing import files. (dce / sad)
Setting up registry information. (dce / sad)
Verifying that the necessary Organization exists. (dce / sad)
Creating group objects from group file. (dce / sad)
```

As **passwd_import** reads the UNIX group and password files, it informs you of any conflicts and prompts for their resolution.

Examining the Group Entries

The **passwd_import** command checks the group file for name and then UNIX ID conflicts. When you resolve the conflict by answering the prompt, **passwd_import** creates the groups in the registry if it is so directed.

The following steps show how group conflicts are handled:

1. The **passwd_import** command first finds a conflict on UNIX IDs. The name **wheel** in the group file and the name **system** in the registry both have UNIX IDs of **0**. The **passwd_import** command prompts you for how to resolve the conflict, as follows:

   ```
   CONFLICT: (wheel 0) - Import Group's UNIX id exists in registry.
             (dce / sad)
             (system 0) is the conflicting entry from the registry.
   Do you wish to resolve the conflict (y) or skip this entry (n):
   ```

2. If you enter an **n** to skip the entry, **passwd_import** continues processing. If you enter a **y** to resolve the conflict, **passwd_import** prompts you for how to resolve the conflict. In the following example, the conflict is resolved by creating an alias of **wheel** for the **system** entry in the registry.

```
Do you wish to resolve the conflict (y) or skip this entry (n): y
Select one of: (a)lias, (g)enerate, (e)nter, (s)kip entry, (h)elp: a
>>Adding Group entry for: wheel 0
```

Because it is running in verbose mode, **passwd_import** describes the actions it is taking. Each action description is prefaced with the >> (redirection symbols).

If you are not running **passwd_import** in check mode, you are not prompted to resolve the conflict. Instead, you are informed of the conflict and processing continues. The message looks like the following display:

```
CONFLICT: (wheel 0) - Import Group's UNIX id exists in registry.
          (dce / sad)
          (system 0) is the conflicting entry from the registry.
Would need new UNIX id to resolve conflict.  (dce /sad)
```

3. If **passwd_import** does not find conflicts, it displays the group entires as it processes them and, because it is running in verbose mode, the actions it is taking. In the following example, you are not prompted to resolve the name conflict because **passwd_import** was invoked with the **–i** option.

```
CONFLICT: (tgroup 35)
        - Group name exists in registry and UNIX ids match.
          (dce / sad)
>>Import Group: - Ignoring name conflict, as instructed
          (dce / sad)
>>Adding Group entry for: diags 48
>>Adding Group entry for: cheetah 50
>>Adding Group entry for: mkt_dev 52
```

As **passwd_import** continues through the UNIX group file, it finds two other UNIX ID conflicts — UNIX entries **adm 4** and **rje 8**, which are in conflict with registry entries **locksmith 4** and **server 8**, respectively.

Examining the Password File

The **passwd_import** command then proceeds to examine the password file for conflicts. As it begins, it displays the following:

```
Creating principal entries and accounts from password file.
         (dce / sad)
```

When an entry is processed with no conflicts, **passwd_import** creates the principal in the registry, adds the principal to the appropriate group and organization, and creates an account for the principal. As it does this, it displays the following:

```
>>Adding Principal entry for: rje
>>Adding account for rje none.
```

The following example shows the warning message that is displayed when **passwd_import** finds a conflict:

```
CONFLICT: (bin 2)
        - Principal name exists in registry and UNIX ids match.
          (dce / sad)
>>Import Principal: - Ignoring name conflict, as instructed
          (dce / sad)
```

This message notifies you that the account for **bin** exists in the registry. Both accounts remain unchanged even though the UNIX password file entry contains information that differs from the registry account.

Because the **–i** option is specified in the command used in the sample session, name conflicts are ignored. The following example shows the prompt from a name conflict that was found when **passwd_import** was run without the **–i** option:

```
CONFLICT: (daemon 1)
        - Principal name exists in registry and UNIX ids match.
          (dce / sad)
Do you wish to resolve the conflict (y) or skip this entry (n):
```

If you enter an **n**, the entry is skipped and processing continues. If you enter a **y**, **passwd_import** prompts you for the new name for the foreign principal, as follows:

```
Enter new name for principal "daemon" "1":
```

Adding Members to Groups

When **passwd_import** completes the processing of the UNIX password file, it re-examines the group file and adds the newly created principals to any additional groups that it finds there. As it does, it displays the following:

```
Add memberships from imported group file. (dce / sad)
>>Add root as member of group with UNIX id: 0
>>Add root as member of group with UNIX id: 2
>>Add daemon as member of group with UNIX id: 2
```

Completing Processing

When **passwd_import** completes processing, it displays the following:

```
Closing import files. (dce / sad)
Closing connection to registry. (dce / sad)
```

Examples — Sample passwd_import Session

Registry Group and Password Entries

- **Group Entries:**

```
wheel::0:
daemon::1:
none::2:
backup::3:user
locksmith::4:
login::5:
mail::6:bin
bin::7:root
server::8:
sys::9:root
staff::10:
sys_admin::11:user
sys_proj::12:
tgroup::35:
```

- **Password Entries:**

```
root:sqlRclUrrblL6:0:10::/:
daemon:sqlRc1UrrblL6:1:2::/:
none:sqlRclUrrblL6:2:2::/:
user:sqlRclUrrblL6:3:2::/:
lp:sqlRclUrrblL6:4:7::/:
sys_person:sqlRclUrrblL6:5:2::/:
admin:sqlRclUrrblL6:6:2::/:
uucp:sqlRclUrrblL6:7:2::/usr/spool/uucppublic:
bin:sqlRclUrrblL6:8:7::/:
```

AIX Group and Password File Entries

- **Group File Entries:**

```
root::0:root
other::1:
bin::2:root,bin,daemon
sys::3:root,bin,sys,adm
adm::4:root,adm,daemon
mail::6:root
rje::8:rje,shqer
daemon::12:root,daemon
tgroup::35:
diags::48:brown,smith,jones
cheetah::50:root,daemon
mkt_dev::52:roberts,anderson,hill
```

- **Password File Entries:**

```
root::0:1:0000-Admin(0000):/:
daemon::1:1:0000-Admin(0000):/:
bin::2:2:0000-Admin(0000):/bin:
sys::3:3:0000-Admin(0000):/usr/src:
adm::4:4:0000-Admin(0000):/usr/adm:
uucp::5:5:0000-uucp(0000):/usr/lib/uucp:
nuucp::10:10:0000-uucp(0000):/usr/spool/uucppublic:\
/usr/lib/uucp/uucico
rje::18:18:0000-rje(0000):/usr/rje:
trouble::70:1:trouble(0000):/usr/lib/trouble:
lp::71:2:0000-lp(0000):/usr/spool/lp:
setup::0:0:general system administration:/usr/admin:/bin/rsh
powerdown::0:0:general system administration:/usr/admin:/bin/rsh
sysadm::0:0:general system administration:/usr/admin:/bin/rsh
checkfsys::0:0:check diskette file system:/usr/admin:/bin/rsh
makefsys::0:0:make diskette file system:/usr/admin:/bin/rsh
mountfsys::0:0:mount diskette file system:/usr/admin:/bin/rsh
umountfsys::0:0:unmount diskette file system:/usr/admin:/bin/rsh
```

Security Object Names

Introduction — Security Object Names

Because the Security namespace is rooted in the Cell Directory Service (CDS) namespace, Security objects have CDS path names, which take the following form:

/.../cellname/mount_point/object_name/

where:

cellname Is the name of the cell in which the object resides.

mount_point Is the name under which the Security Service is registered in the CDS.

object_name Is the name of the registry object assigned when the object is created. If the object resides in a directory, *object_name* consists of the names of the object itself and any directories that must be traversed to access the object. Note that registry objects generally reside in the principal, group, or organization directory in the registry database.

More Detailed Information — Security Object Names

Using Names with dcecp Security Commands

For all **dcecp** commands used to manage the Security Service, except **dcecp acl**, you supply only an object name to identify the object you want to manipulate. The object names are stored in the registry database. You are not required to enter a cell name (the local cell is assumed) or mount point (the name registered for the Security Service is assumed).

Using Names with the dcecp acl Command

Unlike other Security commands, the **dcecp acl** command works with ACLs that can be maintained by DCE services other than Security. Like any generic tool that operates on objects that can exist in different namespaces, **dcecp acl** requires the object's fully-qualified path name instead of just *object_name*.

Note also that, to use **dcecp acl** to manipulate the ACL on the principal directory of the registry database and thus control who can add or delete principals, you must enter the following fully-qualified name:

```
/.../dresden.com/sec/principal
```

In a hierarchical cell, one name can represent a directory and a principal. For example, a principal name stored in Cell A's registry represents a cell with which Cell A engages in cross-cell authentication. The name for the cell in the registry is:

```
/.:/sec/principal/vienna.com
```

This cell name can also represent the name of a directory, such as:

```
/.:/sec/principal/vienna.com/violinists_cell
```

For these cases, the **dcecp acl** command provides an option that identifies whether you are entering a directory name or a principal name.

Examples — Security Object Names

- The full path name for the principal **bach** (which resides in the cell **dresden.com**, uses the **sec** (Security) mount point, and is in the **principal** directory) is as follows:

  ```
  /.../dresden.com/sec/principal/bach
  ```

- As another example, assume the group **east-west** resides in **sales**, which is a subdirectory of the **group** directory in the registry database in the **dresden.com** cell. The full path name for **east-west** is:

  ```
  /.../dresden.com/sec/group/sales/east-west
  ```

- To use the **dcecp acl** command to change the ACL associated with principal **bach**'s registry account, you must enter the fully-qualified name:

  ```
  /.../dresden.com/sec/principal/bach
  ```

  OR

  ```
  /.../dresden.com/sec/principal
  ```

---

sec_admin Program Usage

Introduction — sec_admin Program Usage

While the DCE control program includes commands that create and maintain the primary and secondary replicas of the Registry, the program does not provide functions that are useful for reconfiguring the replica set in a cell. Reconfiguring the Registry replica set in a cell involves reassigning the roles of the primary and secondary replicas. This reassignment of roles typically accompanies a change in the operations on one or more of the nodes in the cell. It becomes desirable for some reason to change the site of the primary replica, and the easiest way to move the primary is to swap its role with a secondary replica. The **sec_admin** program commands that you use to exchange primary and secondary replica roles are the **become** command and **change_master** command. The following sections provide instructions for starting, stopping, and getting help on the Registry Editor, and provide brief descriptions of the **become** and **change_master** commands.

More Detailed Information — sec_admin Program Usage

The following list provides brief descriptions of the **sec_admin** commands for reconfiguring the Registry's replica set. Instructions for changing primary and secondary replica roles can be found in "Troubleshooting Procedures" on page 1-58.

Name	Description
b[ecome_master]	Causes a secondary replica to assume the role of the primary replica.
c[hange_master]	Causes the current primary replica to give its role to a secondary and to become a secondary replica.

For detailed descriptions of the **sec_admin** commands, refer to the *DCE for AIX Administration Command Reference*.

Steps — sec_admin Program Usage

- To start the **sec_admin** program, enter the following command:

  ```
  $ /opt/dcelocal/bin/sec_admin
  ```

  When it is invoked, **sec_admin** enters the interactive mode in which you can execute commands.

- To exit from **sec_admin**, enter the following command:

  ```
  sec_admin> quit
  ```

- To display a list of the **sec_admin** commands, enter the following command:

  ```
  sec_admin> help
  ```

Server Accounts

Introduction — Server Accounts

Servers, which can also be called *applications*, that engage in communications across the network can run under their own network identity or the network identity of the principal who started them. To run under their own identity, servers must be programmed to perform login and then authenticate that identity. Therefore, you must use the **dcecp account create** command to create registry accounts for these servers.

More Detailed Information — Server Accounts

During login, all principals (human, server, and machine) must submit their password to the Authentication Service, which uses these passwords to generate authentication keys. The most common method for human users is to simply enter their password. A different method must be provided for server principals. The recommended method, which is based on APIs supplied with DCE, is to store server keys in a locally protected key table. The default implementation of the DCE-supplied API stores the key table in a keytab file on the server's local machine and protects the file so that only a principal's local identity can read or write the file.

You can access the keytab file remotely. On the local machine, store the keytab files in a partition of the machine's disk that is not exported by any file system.

Except for servers running as **root** or under the identity of the local machine, use a separate keytab file for each server. During login, the server can access this file to obtain its password, pass it to the Authentication Service, log in, and be authenticated.

Use the **dcecp keytab add** command to add passwords for servers to the keytab file and **dcecp keytab remove** command to delete server passwords.

Steps — Server Accounts

To create an account for a server, first run the **dcecp account create** command to create the account and then run the **dcecp keytab add** command to add the server's password to the keytab file. The server's password in the registry and the server's password in the keytab file must match. You can ensure that these passwords are the same by manually entering the same passwords in both subcommands, or you can specify that the **keytab add** command should reset the server's registry password at the same time that it sets the server's password in the keytab file.

Setting DCE Audit Environment Variables

There are three environment variables that are related to the operation of the DCE Audit Service. The DCE Audit environment variables should be set before running the application server (that is, the DCE Audit client). The environment variables are:

DCEAUDITOFF If this variable is declared at the time the application is started, auditing is turned off. By default, this variable is not declared.

DCEAUDITFILTERON If this variable is declared at the time the application is started, filtering is enabled. By default, this variable is not declared, that is, there is no filtering and all audit events are recorded.

DCEAUDITTRAILSIZE Sets the maximum size of the audit trail.

Setting the _s(sec) Variable

Introduction — Setting the _s(sec) Variable

You can supply the name of the registry site to bind to as an argument to the **dcecp** commands that operate on the registry. If you do not supply a name, the command binds to the replica named in the **_s(sec)** variable. If this variable is not set, the command binds to the cell's primary replica.

More Detailed Information — Setting the _s(sec) Variable

The name of the new default replica that you supply as an argument to the **set** command can be in any of the following forms:

- A cell name (for example, /**.../dresden.com**) — If you enter a cell name, the named cell becomes the default cell. The **dcecp** command randomly chooses a replica to bind to in the named cell and that replica becomes the default replica.
- The global name given to the replica when it was created (for example /**.../dresden.com/subsys/dce/sec/rs_server_250_2**) — A global name identifies a specific replica in a specific cell. That cell becomes the default cell and that replica becomes the default replica.
- The replica's name as it displays on the replica list of the current default replica (that is, its cell-relative name, such as **subsys/dce/sec/rs_server_250_2**) — That replica becomes the default replica and the cell in which the replica exists becomes the default cell.
- The network address of the host on which the replica is running (for example, **ncadg_ip_udp:15.22.144.248**) — The replica on that host becomes the default replica and the cell in which the host exists becomes the default cell.

Some of the **dcecp** commands can act only on the primary replica and thus require binding to the primary. If you execute a command that acts only on the primary and the primary is not the default replica, **dcecp** automatically attempts to bind to the primary replica in the current default cell. If this attempt is successful, **dcecp** displays a warning message, informing you that the default replica has been changed to the primary registry. The primary registry will then remain the default replica until you change it. If the attempt to bind is not successful, **dcecp** displays an error message, and the command fails.

Example — Setting the _s(sec) Variable

You can set the **_s(sec)** variable and then use that replica as the default replica for **dcecp registry** commands. To do so, use the **set** command as shown in the following sample that sets the default replica to the primary replica (named **slave_3**) in the cell **giverny.com**:

```
dcecp> set _s(sec) /.../giverny.com/subsys/dce/sec/slave_3
dcecp>
```

Setting Up the Registry

Introduction — Setting Up the Registry

This section describes the steps that you take to set up the registry in the Security Service. Some of these steps are automatically handled by the configuration process during DCE installation and configuration, while others are performed by you using the DCE utilities and control programs.

More Detailed Information — Setting Up the Registry

Because the registry uses the Cell Directory Service to obtain information about network resources, this section assumes that your network is configured properly for Cell Directory Service operation.

For additional information, see:

- “Planning Sites for Security Service Components” on page 7-103
- “Creating the Primary Registry Database” on page 7-56
- “Starting the Master Replica” on page 7-141
- “Populating the New Registry Database” on page 7-106

- "Creating Slave Replicas" on page 7-56
- "Verifying that Replicas are Running" on page 7-146

Steps — Setting Up the Registry

The steps for setting up the registry are as follows:

1. Plan where the Security Service components are to be located in your network.
2. Create the primary registry database (performed by the configuration process).
3. Start the primary replica (performed by configuration process).
4. Populate the registry database (performed by you using the **dcecp** or **rgy_edit** command).
   a. Set policies and properties.
   b. Add names and accounts.
5. Create a secondary database and start the secondary replica (performed by the configuration process).
6. Set up **cron** to run **passwd_export** on all of the DCE-based machines to ensure that the local password and group files are kept consistent with the registry (performed by you using standard AIX commands). The **passwd_export** command is described in "Ensuring Consistent Local Files" on page 7-74.

Starting a Security Server in Locksmith Mode

Use the following form of the **secd** command to start a Security server in locksmith mode:

```
$ /opt/dcelocal/bin/secd [-locksm[ith] pname[-lockpw][rem[ote]]]
```

where:

–locksm[ith] Starts a Security server in locksmith mode.

pname Specifies the name of the locksmith principal. If no registry account exists for this principal, **secd** creates one.

–lockpw Prompts for a new locksmith password. This option allows you to specify a new password for the locksmith account when the old one is unknown.

–rem[ote] Allows the locksmith principal to log in remotely. If this option is not used, the principal must log in from the local machine on which **secd** will be started.

Starting the Audit Daemon

The DCE Audit Service is not a distributed application. The Audit daemon (**auditd**) does not need to run on all DCE hosts even if a client application is making use of the Audit service. The Audit daemon only needs to run on a host if the audit logs are to go to the central trail file or if filters are to be installed on the host. This is because the Audit daemon controls access to the central trail file and also manages the audit filters. However, since the DTS daemon and the Security server daemon are Audit clients, you may want to consider running the Audit daemon on all hosts in the cell.

You must be **root** to be able to start the Audit daemon. Use the following command to start the Audit daemon:

```
$ auditd
```

This command uses flags that influence the behavior of the daemon. For more details on these flags, refer to the *DCE for AIX Administration Command Reference*.

Starting the Master Replica

After configuration creates the master replica, it starts the master replica. To start the master replica (**secd**) explicitly, use the following steps:

1. Log in as **root** on the machine that will run the master replica.
2. Use **ps** to ensure that a **dced** is running on the machine. If one is not, start one. To do so, ensure you are **root** and enter:

   $ **/opt/dcelocal/bin/dced**
3. Start the master replica by entering:

   $ **/opt/dcelocal/bin/secd**

   Set up **secd** so that it starts automatically when the machine is rebooted.

The /etc/passwd and /etc/group Files and the Registry

You should maintain standard versions of the /**etc**/**passwd** and /**etc**/**group** files on local machines to ensure compatibility with AIX programs. To keep the /**etc**/**passwd** and /**etc**/**group** files consistent with the registry database, use the **passwd_export** command. It is advisable to run **passwd_export** on a regular basis, preferably using **cron**. See “Setting the **_s(sec)** Variable” on page 7-138 for details on **passwd_export**.

Note: Unlike standard AIX behavior, the /**etc**/**passwd** and /**etc**/**group** files are not used for local login if a Security server is unavailable. Instead, the local registry (described in “The Local Registry”) is used. The /**etc**/**passwd** and /**etc**/**group** files are maintained *only* for compatibility with AIX programs that require their existence.

The Local Registry

The local registry, which resides in the /**opt**/**dcelocal**/**var**/**security** directory on each local machine, contains information about the machine's most recent users and the date and time that they last logged in. If a Security server is not available for network login, the Authentication Service attempts to obtain the information that is required for a local login from the local registry.

When a Security server is running on the network, the Authentication Service automatically creates a local registry the first time anyone logs into the DCE from the machine. Thereafter, it updates the local registry each time anyone logs into the DCE from the machine. You can edit the local registry by using the **rgy_edit** command with the **–l** flag. Note that the **dcecp** command does not access the local registry. See “Maintaining the Local Registry” on page 7-92 for an example of how to use the **rgy_edit** command to perform this task.

The passwd_import Command Syntax

The **passwd_import** command has the following syntax:

```
/opt/dcelocal/bin/passwd_import [-h] [-c] -d pathname [-i]
                          [-o org] [-p password}
                          {-u username} [-v]
```

where:

–h Displays usage information.

–c Runs in check mode and processes the command showing conflicts, but makes no changes to the registry.

–d *pathname*
Represents path to the directory containing the password and group files to be imported.

–i Indicates identical name strings are not in conflict, but represent the same identity.

–o *org* Represents the name of the registry organization to be assigned to principals added to the registry. The default is the organization named **none**.

–p *password*
The password for the account with whose privileges **passwd_import** will run. If you do not use the **–i** option, **passwd_import** prompts you to resolve the name conflict.

–u *username*
The principal name of the account with whose privileges **passwd_import** will run. This account must have the privileges to access the registry and add principals, groups, accounts and organizations, and members to groups and organizations. The principal name and password are used to obtain network authentication. If you do not supply them, **passwd_import** prompts you for them, even if you have already performed a network login.

–v Runs in verbose mode, generating a verbose transcript of all activity.

The Registry Database

Introduction — The Registry Database

The registry database contains the following information:

Principals Principals are the users of the system. Principals can be interactive (human users) or noninteractive (servers, machines, and cells). Principals can be associated with access permissions.

Groups Groups are collections of principals identified by a group name. Groups can be associated with access permissions.

Organizations Organizations are collections of principals; these principals are identified by an organization name. Organizations define the policies associated with the principals in the registry. Organizations cannot be associated with permissions.

Accounts Accounts contain the passwords and accounting information which allow principals authenticated access to objects within the cell. (Authenticated access can also occur between principals in different cells as described in the following text.)

Policies and Properties Policies and properties regulate such things as password length, password format, and certain authentication requirements.

replist Object Use the **replist** object to manage replicas of the registry database.

xattrschema Object The **xattrschema** object is the extended registry schema created with the ERA facility.

More Detailed Information — The Registry Database

In addition to the directories, the registry contains the **policy** object, the **replist** object, and the **xattrschema** object (all created when the registry is created during machine configuration). The **policy** object contains information that applies to registry properties and policies and to organization policies. The **replist** object contains information about the replicas in the DCE cell. The **xattrschema** object contains information about extended registry attributes. You can modify **policy** and **replica** information at any time using the **dcecp registry** commands. The **xattrschema** object is modified by using the **dcecp xattrschema** commands.

When you create simple objects in the **principal**, **group**, or **org** directory, subdirectories are created as needed. For example, if you add a principal such as **preludes/villa/lobos**, the subdirectories **preludes** and **villa** are created. You can use these subdirectories to help organize your data. When you delete all objects in a subdirectory, the subdirectory itself is deleted. (You cannot delete the **principal**, **group**, or **org** directory.)

The permissions that are granted to objects in the registry depend on where the object fits in the structure of the registry database. Figure 7-8 illustrates the registry database. The boxes represent container objects (directories). The ovals represent simple objects. Figure 7-8 shows only the top level **principal**, **group**, and **org** directories. Your registry can have subdirectories if you create them.

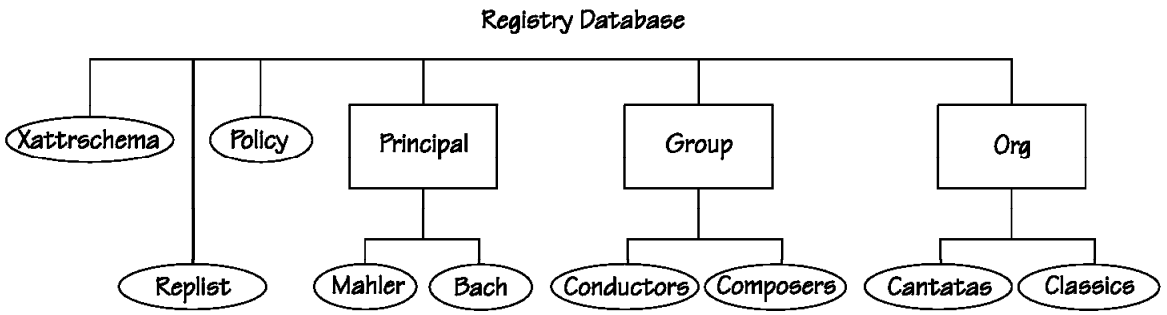


Figure 7-8. The Registry Database Structure

The collection of objects controlled by a registry database is an entity known as a cell. Authenticated communications are possible between cells only if those cells have special accounts in the registry database of the cell with which they want to communicate. These special accounts set up cross-cell authentication, which allows principals from one cell authenticated access to another cell.

The xattrschema Object

Extended attribute types are stored in the object named **xattrschema** under the security junction point (usually **/.:/sec**) in the CDS namespace. Access to the **xattrschema** and the attribute type definitions it contains is controlled by an ACL on the **xattrschema** object. The **xattrschema** object is propagated from the master security server to replicas, like other Registry data.

Trust Relationships

Introduction — Trust Relationships

Note: The DCE for AIX Release 2.1 code does not provide support for transitive trust relationships, which are discussed in this section.

To give explicit permission for principals in other cells to engage in authenticated access to objects in your cell, you must establish a trust relationship with that cell. You do this by using the **dcecp registry connect** command to create two special accounts — one in your cell's registry to represent the foreign cell and one in the foreign cell's registry to represent your cell. Establishing these accounts indicates that you trust the foreign cell's Authentication Service to correctly authenticate foreign users and, therefore, you consider all users from this cell to be authenticated if they are marked as authenticated by the foreign cell's Authentication Service.

More Detailed Information — Trust Relationships

After the trust relationship is established, you can control foreign principals' access to specific objects with ACL entries, just as you do for principals in the local cell. The trust relationship also allows users in the foreign cell to log into accounts in the local cell and vice versa.

Two kinds of trust relationships allow principals in other cells to engage in authenticated access to objects in your cell. These relationships are direct trust relationships and hierarchical transitive trust relationships. Throughout this section the term *transitive trust relationship* is used to indicate the DCE implementation of hierarchical transitive trust relationships.

Direct Trust Relationships

In a direct trust relationship, two cells' Authentication Services share authentication keys and trust each other to authenticate principals from their respective cells. Therefore, both cells consider all users from each cell to be authenticated if they are marked as authenticated by their respective Authentication Service. The shared authentication keys are derived from a single password (one for each cell) that is used by all principals from one cell be authenticated to the other cell. A direct trust relationship involves only two cells.

Transitive Trust Relationships

A transitive trust relationship comes about as a result of a direct trust relationship. In this relationship, cells in a direct trust relationship trust (with some constraints) each other's Authentication Service to authenticate principals not only from their respective cells but also from the cells with which they have direct trust relationships. A transitive trust relationship can involve three or more cells. A transitive trust relationship is illustrated in Figure 7-9.

Example — Trust Relationships

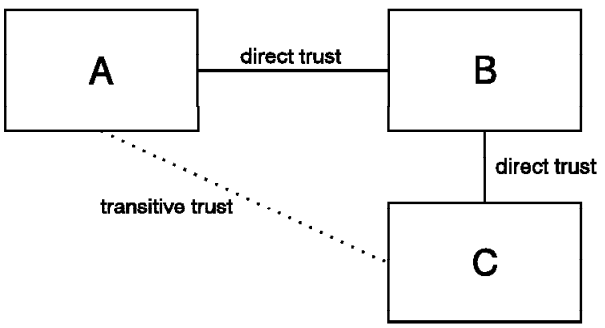


Figure 7-9. Transitive Trust Relationships

In this figure, Cell A trusts peer Cell B (the cell with which it has a direct trust relationship) to authenticate the principals in Cell B and to guarantee the authentication of the principals in Cell B/C (the cell with which it has a transitive trust relationship).

Because Cell A trusts Cell B's Authentication Service, it allows authenticated access to all principals whose authentication is guaranteed by Cell B's Authentication Service. These authenticated principals include principals from Cell B and principals from Cell B/C.

Turning a Master Replica into a Slave

Introduction — Turning Master Replicas into Secondaries

Use this procedure to turn a master replica into a slave. Use this procedure only if you have more than one master running on your network or the Internet, a highly unusual condition.

Steps — Turning Master Replicas into Secondaries

1. Choose the master replica that will become a slave.
2. If necessary, issue the **dcecp registry designate _s(sec)** command to set the chosen master to the default host. The following example sets the default host to /**.../dublin.com/subsys/dce/sec/lit**:

   ```
   dcecp> registry designate _s(sec)/.../dublin.com/subsys/dce/sec/lit:
   ```

3. Issue the following **registry designate –slave** command to change the chosen master to a slave replica:

   ```
   registry designate -slave
   ```

4. Use the **registry show –replica** command to verify the change.

Universal Unique Identifiers and UNIX IDs

Introduction — UUID and UNIX IDs

The Security component automatically associates a principal's, group's, or organization's primary name with a Universal Unique Identifier (UUID). UUIDs identify objects, which is a function performed by UNIX numbers (UNIX IDs) in UNIX systems. (The registry database also contains UNIX numbers, but they are used solely for compatibility with UNIX programs.)

More Detailed Information — UUID and UNIX IDs

Normally, you do not have to be aware of UUIDs. They are created and maintained automatically. However, be aware that while the Security component prints names and you can access objects by name, it identifies all objects internally by UUID. If you delete a principal from the registry, you also delete the principal's UUID. Any objects (files or programs) that are owned by the principal are associated with an orphaned UUID (that is, a UUID with no corresponding name). This means that the object is now owned by a deleted principal. If no other principals were previously given access to the object, the object cannot be accessed.

To solve this problem, use the **dcecp principal create** command with the **–uuid** option to associate the UUID with a name and thus adopt the orphaned object. UUIDs are assigned automatically when the object is created by using the DCE control program's **principal create** command. Therefore, you cannot

simply add a new user and acquire a previously used UUID. You must execute the **dcecp principal create** command with the **–uuid** option for this purpose.

UNIX numbers in the registry must fall within the range of numbers you set as a registry property. When you supply a UNIX number on the command line for creating or modifying an account, avoid numbers like 100, because they are generally reserved for system accounts.

User Accounts

A user account is associated with the user's password and DCE login-related information. Account information includes such things as the principal's home directory and login shell, and authentication policy (which defines parameters that help control a principal's access to the DCE). Use the **dcecp account create** command to create accounts for human users, the **dcecp account modify** command to modify them, and the **dcecp account delete** command to delete them.

Validating the Authenticity of the DCE Security Service

Introduction — Validating Authenticity

The **secval** process within the DCE daemon can confirm that the DCE security server is an authentic server. An illegitimate DCE security server could give a malicious user root access on a machine by returning a counterfeit local system identity. A **secval ping** operation confirms the authenticity of the DCE security server by performing an authenticated RPC to the **secval** process. A successful return (1) indicates that the security server used all of the correct passwords needed for the authenticated RPC to succeed.

More Detailed Information — Validating Authenticity

You can perform a **secval ping** operation on the local host or you can supply an argument to operate on a remote host. Because remote hosts might use different security servers, performing **secval ping** operations on remote hosts provides a way to test the authenticity of other security servers operating in a cell.

Example — Validating Authenticity

The following example illustrates a **secval ping** operation to the **secval** process on remote host **charon**.

```
dcecp> secval ping /.:/hosts/charon/config/secval
1
dcecp>
```

Verifying that Replicas are Running

Introduction — Verifying Replicas are Running

After the primary and secondary replicas are in place and started, perform the steps in this section to ensure the replicas are running.

Steps — Verifying Replicas are Running

1. Invoke **sec_admin**, as follows:

```
$ /opt/dcelocal/bin/sec_admin

Default replica:/.../giverny.com/subsys/dce/sec/art_server_master
Default cell:/.../giverny.com
sec_admin>
```

2. Issue the **lrep** command with the **–state** option to display all Security servers and their status, as follows:

```
sec_admin> lrep -state

Default cell:/.../giverny.com
Default replica:/.../giverny.com/subsys/dce/sec/art_server_master
subsys/dce/sec/art_server_master (master)
          State:              in service - master
          Last update time: Tue Feb  8 14:39:57 1994
subsys/dce/sec/mk
          State:              in service - slave
          Last update time: Tue Feb  8 14:39:57 1994
```

Appendix A. Reference

This section provides reference information that affects various DCE components.

Additional Rules

Table A-1 summarizes major points to remember about CDS, GDS, and DNS character sets, metacharacters, restrictions, case-matching rules, internal storage of data, and ordering of elements in a name. For additional details, see the documentation for each technology.

Table A-1. Summary of CDS, GDS, and DNS Characteristics

Characteristic	CDS	GDS	DNS
Character Set	a to z, A to Z, 0 to 9 plus space and special characters shown in Figure A-8 on page A-49.	a to z, A to Z, 0 to 9 plus .: , ' + - = () ? / and space.	a to z, A to Z, 0 to 9 plus . -
Metacharacters	/ * ? \	/ , = \	.
Restrictions	Simple names cannot begin or end with a slash (/). The first simple name following the global cell name (or /.: prefix) cannot contain an equals sign (=). When entering a name as part of a cdscp show or list command, you must use a backslash (\) to escape any asterisk (*) or question mark (?) character in the rightmost simple name. Otherwise, the character is interpreted as a global file-name character. You can also have an optional character set.	Relative Distinguished Names cannot begin or end with a slash (/). Attribute types must begin with an alphabetic character, can contain only alphanumerics, and cannot contain spaces. An alternate method of specifying attribute types is by object identifier, a sequence of digits separated by dots (.). You must use a backslash (\) to escape a slash (/), a comma (,), and an equals sign (=) when using them as anything other than metacharacters. Multiple consecutive, unescaped occurrences of slashes (/), commas (,), equals signs (=), and backslashes (\) are not allowed. Each attribute value assertion contains exactly one unescaped equals sign (=).	The first character must be alphabetic. The first and last characters cannot be a period (.) or a dash (–). Cell names in DNS must contain at least one dot (.) and they must be more than one level deep.
Case-Matching Rules	Case exact	Attribute types are matched case insensitive. The case-matching rule for an attribute value can be case exact or case insensitive, depending on the rule defined for its type at the DSA.	Case insensitive
Internal Representation	Case exact	Depends on the case-matching rule defined at DSA. If the rule says case insensitive, alphabetic characters are converted to all lowercase characters. Spaces are removed regardless of the case-matching rule.	Alphabetic characters are converted to all lowercase.
Ordering of Name Elements	Big endian (left to right from root to lower-level names)	Big endian (left to right from root to lower-level names)	Little endian (right to left from root to lower-level names)

Criteria for Selecting a Time Source

Introduction — Criteria for Selecting a Time Source

Before you select a time source for your network, ask the following questions:

- **How accurate is the time provided?** Accuracy is affected by the time source itself, as well as the transmission media. As long as the inaccuracy is known, you can compensate for it.
- **How reliable is the time source?** The time source must be available. If the time source is not available, the server connected to the time provider uses times from other servers and compensates for any time difference when the source again becomes available.
- **What is the extent of coverage?** The time source must be available in the geographical area where the time provider server is located.
- **What is the level of known inaccuracy?** If this is known, DTS can compensate for it. Most sources have known inaccuracy levels.
- **What is the cost?**
- **Does the source conform to the operating environment?** The available power supply and physical conditions must be compatible with the source; consult the supplier for specifications.

More Detailed Information — Criteria for Selecting a Time Source

Table A-2 summarizes the selection criteria for each type of time source.

Table A-2. Time Provider Selection Criteria

Selection Criteria			
Type	**Coverage**	**Inaccuracy**	**Cost**
Telephone			
NIST	Regional	10 msec.	Variable fee per call
Radio			
MSF WWV WWVB WWVH	Europe North America North America & Europe Eastern & Central N. Pacific	10 msec. 100 msec. 10 msec. 100 msec.	$1K to 2K $1K to 2K $1K to 2K $1K to 2K
Satellite			
GOES GPS	Worldwide Worldwide	1 msec. corrected < 100 nsec.	$2K to $20K $15K to $20K

Note: The time providers are not supported in DCE for AIX.

DTS Extended BNF

The BNF for DTS time conversion has four parts — **year**, **day**, **tdf**, and **inaccuracy**. For any part whose value is not explicitly expressed, the conversion default value is taken as that of the current day. The BNF for the DTS time conversion is as follows:

```
dts_time :  year_part day_part tdf_part inacc_part
     |   year_part day_part tdf_part
     |   year_part day_part
     |   year_part day_part inacc_part
     |   year_part inacc_part
     |   year_part
     |   day_part  tdf_part inacc_part
     |   day_part  tdf_part
     |   day_part inacc_part
     |   day_part
     |   year_part Z
     |   year_part Z inacc_part
     |   year_part day_part Z inacc_part
     |   day_part  Z inacc_part
     |   day_part  Z
;
year_part : number - number - number -
     | number - number - number T
     | number - number T
     | number T
;
day_part  : partial : partial : partial
     | partial : partial
     | partial
;
tdf_part  : sign number : number
     | sign number
;
sign      : -
     | +
;
partial   : number
     | number frac
     | number frac number
     | frac number
;
frac      : .
     | ,
;
inacc_part : I
     | I partial
     | I infinity
;
infinity   :  'i''n''f'
     |  - -
     |  - - - - -
;
number     :  DIGIT
     |  number DIGIT
;
```

File Locations

Introduction — File Locations

The installation process for DCE for AIX places files in the following locations:

/usr/lpp/dce All DCE files except those in the remainder of this list.

/etc/dce	The following files: **rc.dce** **rc.dts** **dce.clean** **rc.dfs** **dfs.clean** **rpc.clean** **rc.gds** **gds.clean** **rc.dfsnfs** **dce.cf.db** **mkdce.data**
/etc/zoneinfo	Timezone rules for DTS.
/usr/lpp/msg/En_Us	English message catalogs.
/usr/include	Include files (mostly under **/usr/include/dce**)
/usr/ccs/lib	**libc_r.a** and **crt0_r.o**.
/usr/lib	**libdce.a** and **libpthreads.a**.
/usr/lib/drivers	DFS kernel extensions.
/usr/sbin	Commands for loading the DFS kernel extensions.

/opt/dcelocal is set up as a symbolic link to **/usr/lpp/dce**. A link for each of the DCE commands is placed in **/usr/bin**.

In addition, System Management Interface Tool (SMIT) objects are are loaded into the Object Data Manager (ODM) database.

More Detailed Information — File Locations

Files Created at Runtime

The following are files created at runtime:

- **mkdce.data**
- **rc.dts**
- **dce_cf.db**

File Systems to Create

You will probably want to create two new AIX JFS file systems in order to use DCE effectively:

/var/dce	All DCE components store information in the **/var/dce** directory. If the **/var** file system fills up, DCE and other subsystems that depend on **/var** (such as the mail and spooler subsystems) cannot operate correctly. You should create a new file system mounted over **/var/dce** before you install DCE. If you do not plan to create a separate file system for the DFS cache directory, you should add room for the DFS cache to **/var/dce**.
DFS client cache	If the machine will be configured as a DFS client, you should create a new file system to hold the DFS cache files. The default directory is **/var/dce/adm/dfs/cache**, but you can specify a different directory when you configure DFS. The default cache size is 10MB, but you can change this during configuration. If you do not want to create a separate file system, ensure that the

file system where you plan to place the DFS cache has enough room to hold the cache.

Files stored in **/var/dce** are any files particular to the individual machine. These include Security credentials files and the databases for the Security and the CDS servers. You may want to keep an eye on the space usage in **/var/dce** to make sure it does not fill up. To clean up expired credentials files in **/var/dce**, use the **/usr/lpp/dce/bin/rmxcred** command. See the *DCE for AIX Administration Reference* for more information on **rmxcred**.

Maximum Name Sizes

Table A-3 lists maximum sizes for directory service names. Note that the limits are implementation-specific, not architectural.

Table A-3. Maximum Sizes of Directory Service Names

Name Type	Maximum Size
CDS simple name (character string between two slashes)	254 bytes
CDS full name (except Clearinghouse names). CDS uses a 400-byte buffer to contain the fully qualified name. Of this buffer, 27 bytes are reserved for the global prefix (/.../) and each subsequent slash in the name takes 2 bytes.	373 bytes
CDS Clearinghouse name. This is the full name, excluding the global prefix (/.../). The maximum size includes the cell name.	233 bytes
GDS relative distinguished name	64 bytes
GDS distinguished name	1024 bytes
DNS relative name (character string between two dots)	64 bytes
DNS fully qualified name (sum of all relative names)	255 bytes

Metacharacters

Certain characters have special meaning to the directory services and are known as *metacharacters*. Table A-4 on page A-6 lists and explains the CDS, GDS, and DNS metacharacters.

Table A-4. Metacharacters and Their Meaning

Directory Service	Character	Meaning
CDS	/	Separates elements of a name (simple names).
	*	When used in the rightmost simple name of a name entered in a cdscp show or list command, it acts as a global file-name character matching zero or more characters.
	?	When used in the rightmost simple name of a name entered in a cdscp show or list command, it acts as a global file-name character matching exactly one character.
	\	Used where necessary in front of a backslash (\), an asterisk (*) or a question mark (?) to escape the character indicates that the following character is not a metacharacter.
GDS	/	Separates Relative Distinguished Names (RDNs)
	,	Separates multiple attribute type/value pairs (attribute value assertions) within an RDN.
	=	Separates an attribute type and value in an attribute value assertion.
	\	Used in front of slash (/), a comma (,) or an equals sign (=) to escape the character indicates that the following character is not a metacharacter.
DNS	.	Separates elements of a name.

Some metacharacters are not permitted as normal characters within a name. For example, a backslash (\) cannot be used as anything but an escape character in GDS. You can use other metacharacters as normal characters in a name, provided that you escape them with the backslash metacharacter.

Modifying CDS Entity Attributes

Introduction — Modifying CDS Entity Attributes

Every CDS entity has attributes, which are pieces or sets of data that are associated with that entity. Attributes can reflect or affect the operational behavior of an entity, record the number of times a particular event or problem occurred since the entity was last enabled, and uniquely distinguish an entity for any other entity.

CDS attributes are identified by ISO object identifiers (OIDs). Every CDS attribute name maps to an OID and a corresponding data type. Usually, client applications define the name of an attribute and its data type. Application programmers should never need to modify (except for the purpose of foreign language translation) the existing CDS labels associated with the unique OIDs in the **cds_attributes** file. However, programmers can obtain new OIDs from the appropriate authority, create new attributes for their own object entries, and then append them to the existing list. The OID and data type of each attribute are stored in the **cds_attributes** file. Descriptions of the CDS data types that applications can use are in the **cdsclerk.h** file.

More Detailed Information — Modifying CDS Entity Attributes

Adding a New Attribute

Use the **dcecp** program's **modify** command with the **–add** option to add a new attribute to an object entity.

To add a new attribute, you must have previously added the new attribute to the **cds_attributes** file on each host in the cell.

Modifying the Value of an Existing Attribute

Use the **dcecp** program's **modify** command with the **–change** option to modify the value of an existing attribute.

Removing an Attribute

Use the **dcecp** program's **modify** command with the **–remove** option to remove an attribute from an object entity.

Permissions Required — Modifying CDS Entity Attributes

- You must also have write permission to the entity to which you are adding new attributes.
- To modify the value of an attribute, you must have write permission to the name whose attributes you are modifying.
- To remove an attribute, you must have write permission to the name whose attribute you are removing.

Examples — Modifying CDS Entity Attributes

- The following command adds the single-valued attribute (**owner**) to a directory (/**.:/admin**) and assigns a value of **Leland** to the new attribute:

  ```
  dcecp> directory modify /.:/admin -add {owner Leland}
  ```

- The following command adds a new multivalued attribute (**vegetables**) to an object (/**.:/admin**/**garden**) and assigns values of **carrots** and **lettuce** to the new attribute:

  ```
  dcecp> object modify  /.:/admin/garden -add {vegetables {carrots} {lettuce}}
  ```

- The following command changes the value of the owner attribute of the /**.:/admin** directory from **Leland** to **Peters**:

  ```
  dcecp> directory modify  /.:/admin -change {owner Peters}
  ```

- To remove an attribute, use the **modify** command with the **–remove** and **–types** options. For example, the following command removes the **owner** attribute from the /**.:/admin** directory:

  ```
  dcecp> directory modify /.:/admin -remove owner -types
  ```

- To remove a single value from a multivalue attribute, use the **–remove** option and specify the value to be removed. For example, the following command removes the **carrots** value of the **vegetables** attribute from the /**.:/admin**/**garden** object:

  ```
  dcecp> object modify  /.:/admin/garden -remove {vegetables carrots}
  ```

Modifying the cds_attributes and cds_globalnames Files

When a programmer develops an application that uses the Directory Service, the directory service manager or the application developer needs to obtain unique identifiers for any new CDS attribute names or GDS attribute types that the new application uses and then update the appropriate file.

If the application stores names in CDS, edit the **cds_attributes** file. Refer to the **cdsclerk.h** file for the list of appropriate data type descriptors. If the application stores names in GDS, edit the **cds_globalnames** file and use the appropriate ASN.1 identifiers to describe the data type, syntax, and case-matching rules for the name.

Note: If you modify the OID values for standard attributes in the **cds_attributes** and **cds_globalnames** files, you may encounter problems interoperating with other Directory Service implementations.

Origin of Object Identifiers

Introduction — Origin of Object Identifiers

The purpose of object identifiers is to ensure uniqueness among the attribute types that many different applications generate and use. Object identifiers are typically obtained from a hierarchy of allocation authorities, the highest being the International Organization for Standardization (ISO) and the International Telegraph and Telephone Consultative Committee (CCITT).

More Detailed Information — Origin of Object Identifiers

Individual application developers do not usually have to contact ISO or CCITT directly to obtain unique numbers. Application developers are more likely to request object identifiers from a person within their company who is in charge of allocating them. The company authority would, in turn, contact a higher authority to obtain a unique company prefix.

The hierarchy of allocation authorities is indicated by dots that separate portions of an object identifier. Each string of numbers delineated by dots represents a level of the allocation hierarchy, going left to right from the highest authority down.

Example — Origin of Object Identifiers

The sample object identifier 1.3.22.1.1.2 consists of the following levels:

1 ISO

3 Identified organization

22 Open Software Foundation

1 Distributed Computing Environment

1 Remote Procedure Call

2 RPC Object UUIDs

Sources of Coordinated Universal Time

Introduction — Sources of UTC

There are several sources of UTC time including telephone, radio, and satellite, as described in the following sections.

More Detailed Information — Sources of UTC

Telephone Services

Telephone time provider services require the time provider to dial a centralized UTC time source through a modem. Modem speeds and line delays can affect the accuracy of the time returned.

Telephone services are usually provided by standards agencies. For example, in the United States this service is offered by the National Institute of Standards and Technology (NIST). There is a per-call fee for the service in addition to the cost of the modem software.

Radio Transmissions

DTS can obtain time from a radio time source. Commercial receivers that monitor time and frequency broadcasts can return time values back through the TPI to the DTS server. The NIST operates the following U.S. time and frequency stations:

WWV Transmits at 2.5, 5, 10, 15 MHz to North America and South America.

WWVB Transmits at 60 kHz primarily to the United States, providing high-quality frequency information because atmospheric propagation effects are relatively minor.

WWVH Transmits at 2.5, 5, 10, 15 MHz to Alaska, Hawaii, Australia, New Zealand, Japan, and Southeast Asia.

The following stations are available in Europe:

MSF Broadcasts from England at 60 kHz.

DCF77 Broadcasts from Germany at 77.5 kHz.

In addition to the stations listed above, more than 30 radio stations worldwide provide UTC time. Consult the national standards organization in your country for further information.

Network Time Protocol

Nodes that have Internet access can use the Network Time Protocol (NTP) as a source of UTC time for DTS. See “Interoperating with Network Time Protocol” on page 1-42, for an explanation of how to use NTP as a time provider.

Satellite

Satellites have worldwide availability; they can provide relatively precise times if you know their delay and compensate for them. See the following list for satellite sources of UTC:

GOES Geostationary Operational Environment Satellite

TRANSIT A U.S. Navy satellite system consisting of four tracking systems and two ground satellite communication sites

GPS Global Positioning System, a satellite receiver

The cds_attributes File

Introduction — The cds_attributes File

The **cds_attributes** file contains object identifiers for CDS attributes and object classes.

More Detailed Information — The cds_attributes File

Application programmers should never need to modify (except for the purpose of foreign language translation) the CDS labels associated with the unique object identifiers (OIDs) in the **cds_attributes** file. However, programmers can obtain new OIDs from the appropriate allocation authority, create new attributes for their own object entries, and then append them to the existing list.

Example — The cds_attributes File

The following is a sample portion of the default contents of the file:

```
#     OID       LABEL                      SYNTAX
#
1.3.22.1.3.10   CDS_Members             GroupMember
1.3.22.1.3.11   CDS_GroupRevoke         Timeout
1.3.22.1.3.12   CDS_CTS                 Timestamp
1.3.22.1.3.13   CDS_UTS                 Timestamp
1.3.22.1.3.15   CDS_Class               byte
1.3.22.1.3.16   CDS_ClassVersion        Version
1.3.22.1.3.17   CDS_ObjectUUID          uuid
1.3.22.1.3.19   CDS_Replicas            ReplicaPointer
1.3.22.1.3.20   CDS_AllUpTo             Timestamp
1.3.22.1.3.21   CDS_Convergence         small
1.3.22.1.3.22   CDS_InCHName            small
1.3.22.1.3.23   CDS_ParentPointer       ParentPointer
1.3.22.1.3.24   CDS_DirectoryVersion    Version
1.3.22.1.3.25   CDS_UpgradeTo           Version
1.3.22.1.3.27   CDS_LinkTarget          FullName
1.3.22.1.3.28   CDS_LinkTimeout         Timeout
1.3.22.1.3.30   CDS_Towers              byte
1.3.22.1.3.32   CDS_CHName              FullName
1.3.22.1.3.34   CDS_CHLastAddress       byte
1.3.22.1.3.36   CDS_CHState             small
1.3.22.1.3.37   CDS_CHDirectories       CHDirectory
1.3.22.1.3.40   CDS_ReplicaState        small
1.3.22.1.3.41   CDS_ReplicaType         small
1.3.22.1.3.42   CDS_LastSkulk           Timestamp
1.3.22.1.3.43   CDS_LastUpdate          Timestamp
1.3.22.1.3.44   CDS_RingPointer         uuid
1.3.22.1.3.45   CDS_Epoch               uuid
1.3.22.1.3.46   CDS_ReplicaVersion      Version
1.3.22.1.3.48   CDS_NSCellname          char
1.3.22.1.3.52   CDS_GDAPointers         gdaPointer
1.3.22.1.3.53   CDS_CellAliases         GroupMember
1.3.22.1.3.54   CDS_ParentCellPointers  ReplicaPointer
1.3.22.1.1.1    RPC_ClassVersion        byte
1.3.22.1.1.2    RPC_ObjectUUIDs         byte
1.3.22.1.1.3    RPC_Group               byte
1.3.22.1.1.4    RPC_Profile             byte
1.3.22.1.1.5    RPC_Codesets            byte
1.3.22.1.5.1    SEC_RepUUID             byte
```

The first column contains the object identifier (OID), the second column contains a label (the name to which the identifier is mapped), and the third column indicates the data type. Descriptions of the CDS data types are in the **cdsclerk.h** header file. See *DSS for AIX Up and Running!* for the full path names of all CDS files.

The cds_globalnames File

Introduction — The cds_globalnames File

The **cds_globalnames** file contains a copy of data that is stored in a Directory Service Agent (DSA) schema for use by GDS. CDS uses this file to interpret the GDS portion of global names that it handles. The file contains only naming attributes (that is, attributes that constitute a distinguished name).

More Detailed Information — The cds_globalnames File

The **cds_globalnames** file contains additional comments and descriptive information about attribute types and case-matching rules. For details on the ASN.1 identifiers and their meaning, see the X.500 recommendation.

Example — The cds_globalnames File

The following is a sample portion of the **cds_globalnames** file:

```
# OID           LABEL   ASN.1-IDENTIFIER        SYNTAX        MATCHING
#
# Reference: X.520 (Selected Attribute Types for the Directory)
2.5.4.0         OC      objectClass             -              -
2.5.4.1         AO      aliasedObjectName       -              -
2.5.4.2         KI      knowledgeInformation    CIS            CIM
2.5.4.3         CN      commonName              CIS            CIM
2.5.4.4         S       surname                 CIS            CIM
2.5.4.5         SN      serialNumber            PS             PM
2.5.4.6         C       countryName             PS             CIM
2.5.4.7         L       localityName            CIS            CIM
2.5.4.8         SP      stateOrProvinceName     CIS            CIM
2.5.4.9         SADR    streetAddress           CIS            CIM
2.5.4.10        O       organizationName        CIS            CIM
2.5.4.11        OU      organizationalUnitName  CIS            CIM
2.5.4.12        T       title                   CIS            CIM
2.5.4.13        D       description             CIS            CIM
#2.5.4.14       SG      searchGuide             Guide          -
2.5.4.15        BC      businessCategory        CIS            CIM
#2.5.4.16       POST    postalAddress           PostalAddress   -
2.5.4.17        PC      postalCode              CIS            CIM
2.5.4.18        POB     postOfficeBox           CIS            CIM
                              ⋮
```

The first column contains the object identifier and the second column contains the string name to which it is mapped. The third column is the ASN.1 identifier for the attribute type, as defined in the appropriate CCITT recommendation (X.500 or X.400). The fourth column is the ASN.1 label for the syntax of the attribute type. The fifth column contains the ASN.1 identifier of the matching rule to be applied to the attribute type. Possible syntax abbreviations are as follows:

CES Case Exact String

CIS Case Ignore String

PS Printable String

NS Numeric String

– Unspecified

Matching rules are defined as follows.

CEM **Case Exact String Matching** — Leading and trailing spaces are ignored and multiple consecutive internal spaces are reduced to one; otherwise, the strings must be the same length and corresponding characters must be identical.

CIM **Case Ignore String Matching** — Same as CEM, except that characters differing only in case are considered to match.

PM **Printable String Matching** — Same as CEM.

NM **Numeric String Matching** — Same as CEM, except that all spaces are ignored.

– Unspecified

The CDS Space

Introduction — The CDS Space

In "More Detailed Information — The CDS Space" on page A-13, the **CDS Class** field is used internally by the **CDS_Clearinghouse** entry and the RPC NSI. The **Well Known** field specifies whether the last component of a name is an architecturally required name. The **Default ACLs** field specifies the ACL created by configuring DCE using **mkdce** or **SMIT**. The **Created by** field specifies how this entry is created.

The *hostname*, *dce_hostname*, *cellname*, and *creator* entries are defined as follows:

hostname This is a cell-relative DCE host name. For example, the *hostname* for a host named **machine1.abc.com** is **machine1**. Note that for cells with subdomains, a directory structure is possible. For example, the host **apollo.mercury.acs.cmu.edu** can have a *hostname* of **acs**/**mercury**/**apollo**.

dce_hostname This is a name assigned by the cell administrator to be the DCE host name of a machine in a cell (the name found in /**.:**/*dce_hostname* entries). A clearinghouse created with a CDS server will have the default name of *dce_hostname*_**ch**.

Note: The *dce_cellname* is case sensitive and should be specified as it was defined at DCE configuration time.

cellname This is the global name of the cell, without the special character string /**...**/ (for example, **seattle.abc.com** or **C=US**/**O=ABC**/**OU=Seattle**).

creator This is the name of the principal that created the cell.

Figure A-1, Figure A-2 on page A-13, and Figure A-4 on page A-13 illustrate the CDS space within the DCE cell namespace. The sections that follow provide a description of each entry.

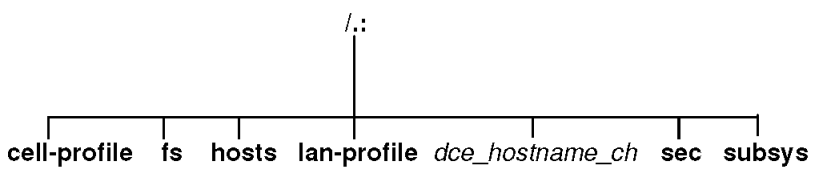


Figure A-1. The Top-Level CDS Directory

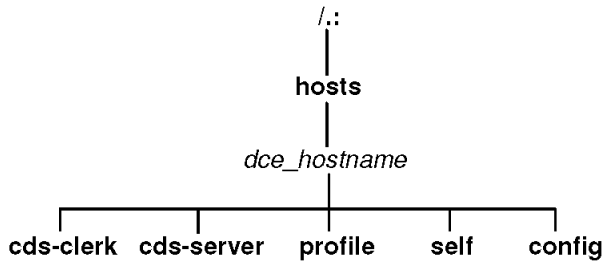


Figure A-2. The CDS hosts Directory

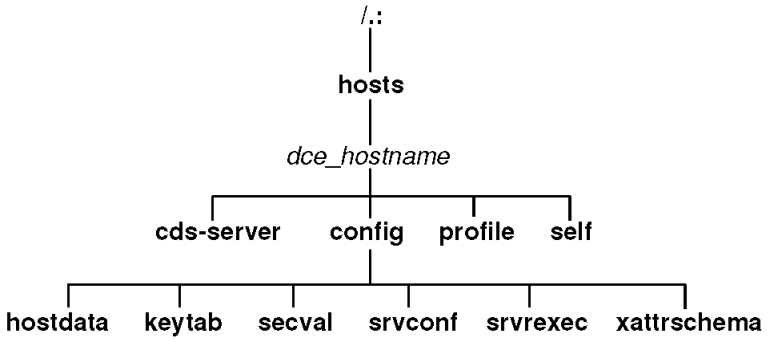


Figure A-3. The Host Daemon Directory

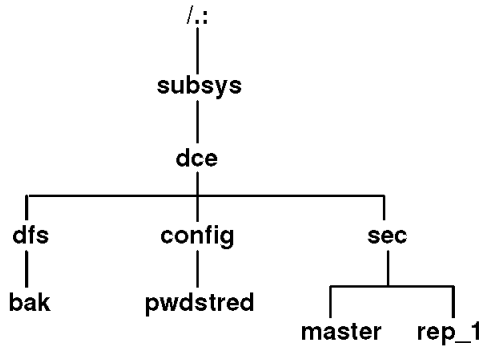


Figure A-4. The CDS subsys Directory

More Detailed Information — The CDS Space

The Top-Level CDS Directory

The following describes the namespace entries for /**.:**, the top-level CDS directory.

Name /**.:**

CDS Type

Directory

Well Known

Yes

Description

This is the cell root directory. The special character string /**.:** is a shorthand form of /**.../***cellname*. This directory is replicated in every clearinghouse.

Default ACLs

Object ACL
Default cell = /**...**/*cellname*
unauthenticated:r----t-
group:subsys/dce/cds-admin:rwdtcia
group:subsys/dce/cds-server:rwdtcia
any_other:r----t-

Def_object ACL
Default cell = /**...**/*cellname*
unauthenticated:r---t
group:subsys/dce/cds-admin:rwdtc
group:subsys/dce/cds-server:rwdtc
any_other:r---t

Def_container ACL
Default cell = /**...**/*cellname*
unauthenticated:r----t-
group:subsys/dce/cds-admin:rwdtcia
group:subsys/dce/cds-server:rwdtcia
any_other:r----t-

Created By
CDS configuration

Name **cell-profile**

CDS Type
Object

CDS Class
RPC-Profile

Well Known
Yes

Description
This is the master default profile for the cell. Ultimately, all host, user, and other profiles must chain up to this profile. This profile is created at cell creation and must include the following entry:

LAN-Services-UUID /.../*cellname***/lan-profile**

Note that like all profile entries, only global names can be used. This profile must include interfaces for the Privilege Server, the Registry Server, and the Authentication Server. All of these interfaces point to /**.:/sec**. (In multi-LAN cells this is the profile in which the time global set entries are entered.)

Default ACLs

Object ACL
unauthenticated:r--t-
user:*creator***:rwdtc**
group:subsys/dce/cds-admin:rwdtc
group:subsys/dce/cds-servers:rwdtc
group:subsys/dce/dts-admin:rw-t-
group:subsys/dce/dts-servers:rw-t-
any_other:r--t-

Created By
DCE configuration

Name /**.:/fs**

CDS Type

Object

CDS Class

RPC_Group

Well Known

No

Description

The RPC bindings of all Fileset Database machines housing the FLDB are listed in this group. This group consists of names of the following form:

/.../*cellname***/hosts/***dce_hostname***/self**

This object needs to have a single object UUID attached to it. This is the junction to the DFS filespace within the cell namespace. **/:** is a CDS soft link to /**.:/fs**.

Default ACLs

Object ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t-
user:*creator***:rwdtc**
group:subsys/dce/cds-admin:rwdtc
group:subsys/dce/cfs-servers:rwdtc
group:subsys/dce/dfs-fs-servers:rwdtc
group:subsys/dce/dfs-admin:rwdtc
any_other:r--t-

Created By

DCE configuration

Name /**.:/hosts**

CDS Type

Directory

Well Known

No

Description

Host directories are catalogued here.

Default ACLs

Object ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t---
user:*creator***:rwdtcia**
user:/*dce_hostname*/**cds-server:rwdtcia**
user:/*dce_hostname*/**self:rwdtcia**
group:subsys/dce/cds-admin:rwdtcia
group:subsys/dce/cds-server:rwdtcia
any_other:r--t---
Def_object ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t---
group:subsys/dce/cds-admin:rwdtc--
group:subsys/dce/cds-server:rwdtc--
any_other:r--t---

Def_container ACL

Default cell = /**...**/*cellname*
unauthenticated:r--t---
group:subsys/dce/cds-admin:rwdtcia
group:subsys/dce/cds-server:rwdtcia
any_other:r--t---

Created By

DCE configuration

Name /**.:/lan-profile**

CDS Type

Object

CDS Class

RPC_Profile

Well Known

No

Description

This is the default LAN profile used by DTS (and potentially other services). In single-LAN cells, this is the profile in which entries for the time service local set entries are entered.

Default ACLs

Object ACL

unauthenticated:r--t-
user:*creator***:rwdtc**
group:subsys/dce/cds-admin:rwdtc
group:subsys/dce/cds-servers:rwdtc
group:subsys/dce/dts-admin:rwdtc
group:subsys/dce/dts-servers:rwdtc
any_other:r--t-

Created By

DCE configuration

Name /**.:/***dce_hostname*_**ch**

CDS Type

Object

CDS Class

CDS_Clearinghouse

Well Known

No

Description

All clearinghouses are catalogued in the cell root. This name is fixed only for the first CDS server you configure. You can create additional clearinghouses for this servers with any name you choose. Additional CDS servers have clearinghouses named *dce_hostname*_**ch**, by default.

Default ACLs

Object ACL

unauthenticated:r--t-
group:subsys/dce/cds-admin:rwdtc
group:subsys/dce/cds-server:rwdtc
any_other:r--t-

Created By

CDS configuration

Name /.:/**sec**

CDS Type

Object

CDS Class

RPC_Group

Well Known

No

Description

This is the RPC group of all security servers for this cell. It contains the entries /**...**/*cellname*/**subsys**/**dce**/**sec**/**master** and, for example, /**...**/*cellname*/**subsys**/**dce**/**sec**/**rep_1**. This is a junction into the security space.

Default ACLs

Object ACL

Default cell = /**...**/*cellname*
unauthenticated:r--t-
user:*creator***:rwdtc**
user:dce-rgy:rwdtc
user:hosts/**rep_1**_*dce_hostname*/**self rwdtc**
group:subsys/**dce**/**cds-admin:rwdtc**
group:subsys/**dce**/**cds-server:rwdtc**
group:subsys/**dce**/**sec-admin:rwdtc**
any_other:r--t-

Created By

DCE configuration

Name /.:/**subsys**

CDS Type

Directory

Well Known

No

Description

This directory contains directories for different subsystems in this cell. There is only one directory specified below this one: **dce**. It is recommended that companies adding subsystems to DCE conform to the convention of creating a unique directory below **subsys** by using their trademark as a directory name (/**.:/subsys**/*trademark*). These directories are used for storage of location independent information about services. Server entries, groups, and profiles for the entire cell need to be stored in directories below **subsys**.

Default ACLs

Object ACL

Default cell = /**...**/*cellname*
unauthenticated:r--t---
user:*creator***:rwdtcia**
user:hosts/*dce_hostname***:rwdtcia**
group:subsys/**dce**/**cds-admin:rwdtcia**
group:subsys/**dce**/**cds-server:rwdtcia**
any_other:r--t---

Def_object ACL

Default cell = /**...**/*cellname*
unauthenticated:r--t---
group:subsys/**dce**/**cds-admin:rwdtc**
group:subsys/**dce**/**cds-server:rwdtc**
any_other:r--t---

Def_container ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t---
group:subsys/dce/cds-admin:rwdtcia
group:subsys/dce/cds-server:rwdtcia
any_other:r--t---

Created By
DCE configuration

The CDS hosts Directory

The following describes the namespace entries for /**.:/hosts**, the CDS **hosts** directory.

Name /**.:/hosts**/*dce_hostname*
CDS Type
Directory
Well Known
No
Description
Each host has a directory in which RPC server entries, groups, and profiles associated with this host are stored. This is simply a CDS directory. No bindings are present in the directory object itself. Entries exist only beneath the directory.
Default ACLs

Object ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t---
user:*creator***:rwdtcia**
user:hosts*/dce_hostname/***cds-server:rwdtcia**
user:hosts*/dce_hostname/***self:rwdtcia**
group:subsys/dce/cds-admin:rwdtcia
any_other:r--t---
Def_object ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t---
group:subsys/dce/cds-admin:rwdtc--
group:subsys/dce/cds-server:rwdtc--
any_other:r--t---
Def_container ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t---
group:subsys/dce/cds-admin:rwdtcia
group:subsys/dce/cds-server:rwdtcia
any_other:r--t---

Created By
DCE configuration

Name /**.:/hosts**/*dce_hostname*/**cds-clerk**

CDS Type

Object

CDS Class

RPC_Entry

Well Known

No

Description

This entry contains the binding for a CDS clerk. This entry is also used by the ACL editor to manage the ACL interface.

Default ACLs

Object ACL

Default cell = /**...**/*cellname*
unauthenticated:r--t-
user:*creator***:rwdtc**
user:hosts/*dce_hostname*/**self:rw-t-**
group:subsys/dce/cds-admin:rwdtc
group:subsys/dce/cds-server:rwdtc
any_other:r--t-

Created By

DCE configuration

Name /**.:/hosts**/*dce_hostname*/**cds-server**

CDS Type

Object

CDS Class

RPC_Entry

Well Known

No

Description

This entry contains the binding for a CDS server. This entry is also used by the ACL editor to manage the ACL interface.

Default ACLs

Object ACL

Default cell = /**...**/*cellname*
unauthenticated:r--t-
user:*creator***:rwdtc**
user:hosts/*dce_hostname*/**self:rw-t-**
group:subsys/dce/cds-admin:rwdtc
group:subsys/dce/cds-server:rwdtc
any_other:r--t-

Created By

DCE configuration

Name /**.:/hosts**/*dce_hostname*/**profile**

CDS Type

Object

CDS Class

RPC_Entry

Well Known

No

Description

This is the default profile for host *dce_hostname*. It must contain a default which points (possibly indirectly) to **/.:/cell-profile**. Programs should acquire this name from the **dce_cf_profile_name_from_host** function.

Default ACLs

Object ACL

Default cell = /**...**/*cellname*
unauthenticated:r--t-
user:*creator***:rwdtc**
user:hosts/*dce_hostname*/**self:rw-t-**
group:subsys/dce/cds-admin:rwdtc
group:subsys/dce/cds-server:rwdtc
any_other:r--t-

Created By

DCE configuration

Name /**.:/hosts**/*dce_hostname*/**self**

CDS Type

Object

CDS Class

RPC_Entry

Well Known

Yes

Description

This entry contains a binding to the **dced** daemon on host *dce_hostname*. The **dce_cf_binding_entry_from_host** routine returns either the name of this entry when handed a DCE host name or the current host when a DCE host name is not provided.

Default ACLs

Object ACL

Default cell = /**...**/*cellname*
unauthenticated:r--t-
user:*creator***:rwdtc**
user:hosts/*dce_hostname*/**self:rwdtc**
group:subsys/dce/cds-admin:rwdtc
group:subsys/dce/cds-server:rwdtc
any_other:r--t-

Created By

DCE configuration

Name /**.:/hosts**/*dce_hostname*/**config**
CDS Type
Object
CDS Class
RPC_Entry
Well Known
Yes
Description
This is the server entry for the **dced** on the given host. It is also the top of the naming tree for that **dced**. Programs obtain this name by using the cell **dce_cf_dced_entry_from_host**.
Default ACLs

Object ACL
unauthenticated r--t---
user host/*dce_hostname*/**self rwdtc--**
group subsys/dce/cds-admin rwdtc--
group subsys/dce/cds-server rwdtc--
any_other r--t---
Created By
DCE configuration

The Host Daemon Directory

The following describes the **dced** namespace entries for /**.:/hosts**/*dce_hostname*/**config**, the host daemon directory. These are all created by **dced** as part of configuration.

Name /**.:/hosts**/*dce_hostname*/**config**
dced Type
dced object
Well Known
Yes
Description
This is the **dced** server itself.
Default ACLs

Object ACL
user hosts/*dce_hostname*/**self crws**
Created By
DCE configuration

Name /**.:/hosts**/*dce_hostname*/**config/hostdata**
dced Type
dced container
Well Known
Yes
Description
This is the container for **hostdata** objects on the given host.
Default ACLs

Object ACL
user hosts/*dce_hostname*/**self cril**
Initial Object ACL
user hosts/*dce_hostname*/**self cdprw**

Created By
DCE configuration

Name /**.:/hosts**/*dce_hostname*/**config/keytab**
dced Type
dced container
Well Known
Yes
Description
This is the container for **keytab** objects on the given host.
Default ACLs

Object ACL
user hosts/*dce_hostname*/**self cril**
Initial Object ACL
user hosts/*hostname*/**self acdepr**
Created By
DCE configuration

Name /**.:/hosts**/*dce_hostname*/**config/secval**
dced Type
dced object
Well Known
Yes
Description
This is the name of the **secval** service.
Default ACLs

Object ACL
user hosts/*dce_hostname*/**self csux**
Created By
DCE configuration

Name /**.:/hosts**/*dce_hostname*/**config/srvrconf**
dced Type
dced container
Well Known
Yes
Description
This is the container for the configured servers registered with **dced**.
Default ACLs

Object ACL
user hosts/*dce_hostname*/**self cril**
Initial Object ACL
user hosts/*dce_hostname*/**self cdfrwx**
Created By
DCE configuration

Name /**.:/hosts**/*dce_hostname*/**config/srvrexec**
dced Type
dced container
Well Known
Yes
Description
This is the container for the running servers registered with **dced**.
Default ACLs

Object ACL
user hosts/*dce_hostname*/**self cril**
Initial Object ACL
user hosts/*dce_hostname*/**self crws**
Created By
DCE configuration

Name /**.:/hosts**/*dce_hostname*/**config/xattrschema**
dced Type
dced container
Well Known
Yes
Description
This is the container of extended attribute schema definitions.
Default ACLs

Object ACL
user hosts/*dce_hostname*/**self cril**
Initial Object ACL
user hosts/*dce_hostname*/**self crwd**
Created By
DCE configuration

The CDS subsys Directory

The following describes the namespace entries for /**.:/subsys**, the CDS **subsys** directory.

Name /**.:/subsys/dce**
CDS Type
Directory
Well Known
No
Description
This directory contains DCE specific names.
Default ACLs

Object ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t---
user:*creator***:rwdtcia**
user:hosts/*dce_hostname*/**cds-server:rwdtcia**
group:subsys/dce/cds-admin:rwdtcia
group:subsys/dce/cds-server:rwdtcia
any_other:r--t---

Def_object ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t-
group:subsys/dce/cds-admin:rwdtc--
group:subsys/dce/cds-server:rwdtc--
any_other:r--t-
Def_container ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t---
group:subsys/dce/cds-admin:rwdtcia
group:subsys/dce/cds-server:rwdtcia
any_other:r--t---

Created By
DCE configuration

Name **./.:/subsys/dce/dfs**
CDS Type
Directory
Well Known
No
Description
This directory contains all DFS specific names.
Default ACLs

Object ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t---
user:*creator***:rwdtcia**
user:hosts/*dce_hostname***/cds-server:rwdtcia**
group:subsys/dce/cds-admin:rwdtcia
group:subsys/dce/cds-server:rwdtcia
group:subsys/dce/dfs-admin:rwci-ta
any_other:r--t---
Def_object ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t-
group:subsys/dce/cds-admin:rwdtc--
group:subsys/dce/cds-server:rwdtc--
group:subsys/dce/dfs-admin:rwdtc--
any_other:r--t-
Def_container ACL
Default cell = /**...**/*cellname*
unauthenticated:r--t---
group:subsys/dce/cds-admin:rwdtcia
group:subsys/dce/cds-server:rwdtcia
group:subsys/dce/dfs-admin:rwdtcia
any_other:r--t---

Created By
DCE configuration

Name **/.:/subsys/dce/dfs/bak**

CDS Type

Object

CDS Class

RPC_Entry

Well Known

No

Description

The RPC bindings of all Backup Database machines storing the Backup Database are listed in this entry. This entry is similar to the /**.:/fs** group, in that its members are RPC bindings of the form /**.../***cellname*/**hosts**/*dce_hostname*/**self**. In addition, this group needs to have a single object UUID attached to it.

Default ACLs

Object ACL

Default cell = /**.../***cellname*
unauthenticated:r--t-
user:*creator***:rwdtc**
user:*/dce_hostname/***cds-server:rwdtc**
group:subsys/dce/cds-admin:rwdtc
group:subsys/dce/cds-server:rwdtc
any_other:r--t-

Created By

DCE configuration

Name **/.:/subsys/dce/pwd_mgmt/pwdstred**

CDS Type

Object

CDS Class

RPC_Entry

Well Known

No

Description

This object contains binding information for the default password strength server.

Default ACLs

Object ACL

Default cell = /**.../***cellname*
unauthenticated:r--t---
user:cell-admin:rwdtc---
user:pwdstred:rwdt---
any_other:r--t---

Created By

DCE Configuration

Name **/.:/subsys/dce/sec**

CDS Type

Directory

Well Known

No

Description

This directory contains security-specific names.

Default ACLs

Object ACL

Default cell = /**...**/*cellname*
unauthenticated:r--t---
user:hosts/*dce_hostname*/**self:rwdtcia**
user:dce-rgy:rwdtci-
group:subsys/dce/cds-admin:rwdtcia
group:subsys/dce/cds-server:rwdtcia
group:subsys/dce/sec-admin:rwdtcia
any_other:r--t---

Def_object ACL

Default cell = /**...**/*cellname*
unauthenticated:r--t---
user:dce-rgy:rwdt---
group:subsys/dce/cds-admin:rwdtc
group:subsys/dce/cds-server:rwdtc
group:subsys/dce/sec-admin:rwdtc
any_other:r--t---

Def_container ACL

Default cell = /**...**/*cellname*
unauthenticated:r--t---
user:dce-rgy:rwdtci-
group:subsys/dce/cds-admin:rwdtcia
group:subsys/dce/cds-server:rwdtcia
group:subsys/dce/sec-admin:rwdtcia
any_other:r--t---

Created By

DCE configuration

Name /**.:/subsys/dce/sec/master**

CDS Type

Object

CDS Class

RPC_Entry

Well Known

No

Description

This is the server entry for the master security server for this cell. The bindings for the Registry Server, the Privilege Server, and the Authentication Server are exported by the Registry Server to this entry.

Default ACLs

Object ACL

Default cell = /**...**/*cellname*
unauthenticated:r--t-
user:*creator***:rwdtc**
user:dce-rgy:rwdt-
group:subsys/dce/cds-admin:rwdtc
group:subsys/dce/cds-server:rwdtcia
group:subsys/dce/sec-admin:rwc-t
any_other:r--t-

Created By

DCE configuration

Name **/.:/subsys/dce/sec/rep_1**

CDS Type

Object

CDS Class

RPC_Entry

Well Known

No

Description

This is the server entry for a secondary Security Server for this cell. The bindings for the Registry Server, the Privilege Server, and the Authentication Server are exported by the Registry Server to this entry.

Default ACLs

Object ACL

unathenticated r--t-
user dce-rgy rwdt-
user *creator* **rwdtc**
user hosts/*rep_1_hostname*/**self rwdtc**
group subsys/dce/cds-admin rwdtc
group subsys/dec/sec-admin rwdtc
any_other r--t-

Created By

DCE configuration

The Security Space

Introduction — The Security Space

In "More Detailed Information — The Security Space" on page A-29, the **Well Known** field specifies whether the last component of a name is an architecturally required name. The **Default ACLs** field specifies the ACL created by the configuration process.

The *hostname*, *dce_hostname*, *cellname*, and *creator* entries are defined as follows:

hostname This is a cell-relative DCE host name. For example, the *hostname* for a host named **machine1.abc.com** is **machine1**. Note that for cells with subdomains, a directory structure is possible. For example, the host **apollo.mercury.acs.cmu.edu** can have a *hostname* of **acs/mercury/apollo**.

dce_hostname This is a name assigned by the cell administrator to be the DCE host name of a machine in a cell (the name found in /**.:**/*dce_hostname* entries). A clearinghouse created with a CDS server will have the default name of *dce_hostname*_**ch**.

Note: The *dce_cellname* is case sensitive and should be specified as it was defined at DCE configuration time.

cellname This is the global name of the cell, without the special character string /**.../** (for example, **seattle.abc.com** or **C=US/O=ABC/OU=Seattle**).

creator This is the name of the principal that created the cell.

Figure A-5 on page A-28, Figure A-6 on page A-28, and Figure A-7 on page A-29 illustrate the security space within the DCE cell namespace. Following the figures is a description of each entry. The subdirectories that comprise the security space are **principal**, **group**, **org**, **policy**, **replist**, and **xattrschema**.

To operate on the ACLs on any of these namespace entries, you need to include the name of the security junction. For example, when you use the DCE control program's (**dcecp**) **acl *** commands, the group name **acct-admin** is referenced as **/.:/sec/group/acct-admin** (its database object name).

However, the **dcecp** program's **principal ***, **group ***, or **organization *** commands operate on a principal, group, or organization name without including **/.:/sec** and **principal, group,** or **organization** as part of the name. For example, to view the attributes of the group **acct-admin**, issue the **group show** command specifying the group name **acct-admin** without this path.

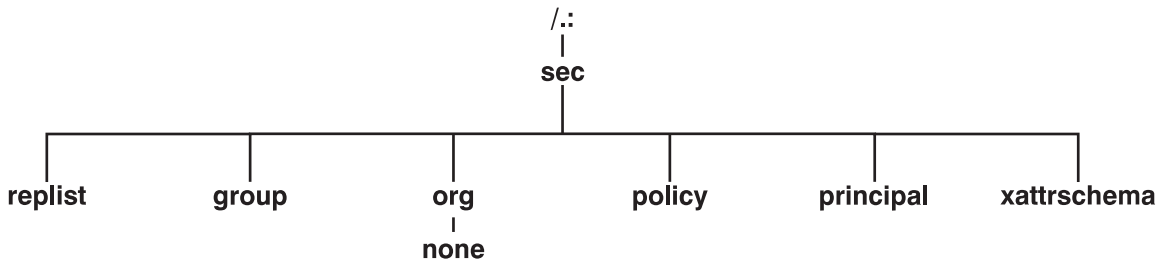


Figure A-5. The Top-Level Security Directory

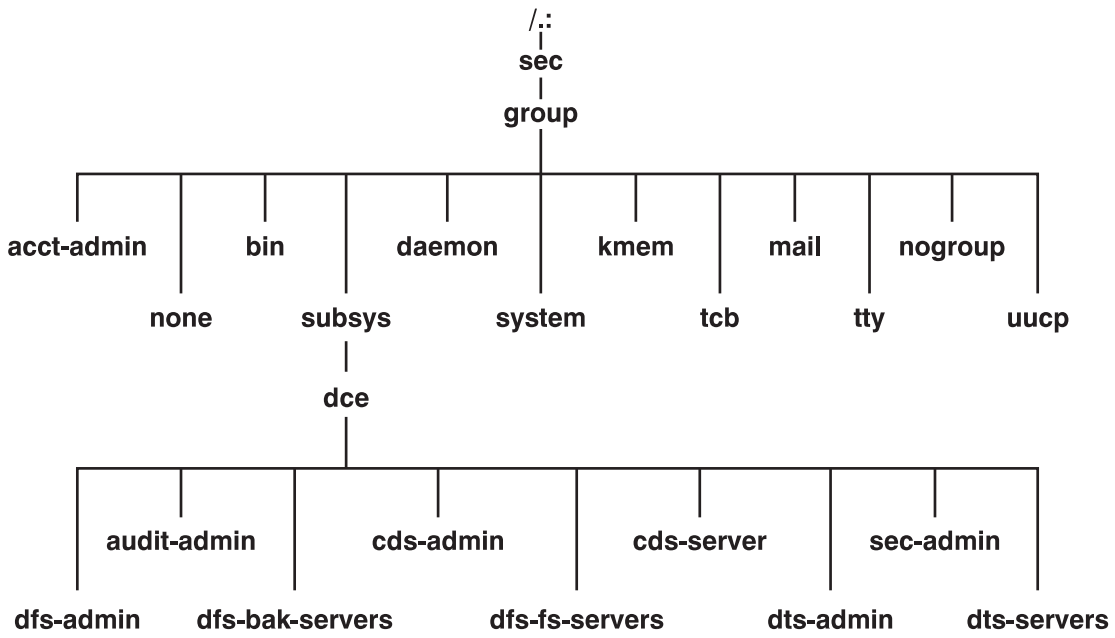


Figure A-6. The sec/group Directory

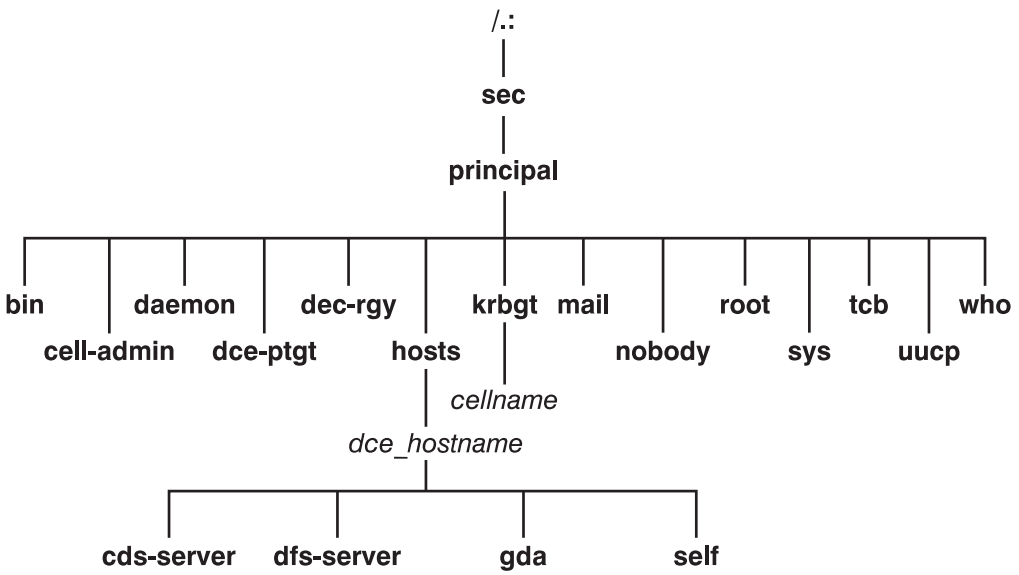


Figure A-7. The sec/principal Directory

In "More Detailed Information — The Security Space," descriptions of entries in an initial security namespace are given. Included is the suggested UNIX user identifier (UNIX UID) or group identifier (UNIX GID) assigned to the entries. Vendors should use these values if possible. The password and group override files can replace them with correct local values, if necessary. Some entries are assigned the next available identifier (starting with 100). Therefore, these may vary from cell to cell. They are indicated as *Generated*.

More Detailed Information — The Security Space

The Top-Level Security Directory

The following describes the namespace entries for /**.:/sec**, the top-level security directory.

Name /**.:/sec/group**

Well Known

Yes. This name is not architecturally defined, but is defined by the implementation.

Description

This is the security directory that holds all the groups.

Default ACLs

Object ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----

Initial Object ACL
unauthenticated r-t----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-------

Initial Container ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----

Created By
Security configuration

Name **/.:/sec/org**

Well Known
Yes. This name is not architecturally defined, but is defined by the implementation.

Description
This is the security directory that holds all of the organizations.

Default ACLs

Object ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----
Initial Object ACL
unauthenticated r-t----
user *creator* **rctDnfmM**
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----
Initial Container ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----

Created By
DCE configuration

Name **/.:/sec/org/none**

Well Known
Yes.

Description
The default organization.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

Created By
Security configuration

Name **/.:/sec/policy**

Well Known

Yes. This name is not architecturally defined, but is defined by the implementation.

Description

This entry provides the ability to set security policies on a cell-wide basis.

Default ACLs

Object ACL
unauthenticated r----
user *creator* **rcmaA**
group acct-admin rcmaA
other_obj r----
any_other r----

Created By

DCE configuration

Name **/.:/sec/principal**

Well Known

Yes. This name is not architecturally defined, but it cannot be changed in DCE for AIX Version 2.1.

Description

This is the security directory that holds all the principals.

Default ACLs

Object ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----
Initial Object ACL
unauthenticated r-------g
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------
Initial Container ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----

Created By

Security configuration

Name **/.:/sec/replist**

Well Known

Yes. This name is not architecturally defined, but it cannot be changed in DCE for AIX Version 2.1.

Description

This entry holds information about the different security replicas.

Default ACLs

Object ACL
user *creator* **cidmA-**
user hosts/*dce_hostname*/**self-i-m-l**
group acct-admin cidmA-

Created By
DCE configuration

Name /**.:/sec/xattrschema**

Well Known
Yes. This name is not architecturally defined, but it cannot be changed in DCE for AIX Version 2.1.

Description
This is a container for extended registry attribute schema entries. The entries within this directory define the format of ERAs that may be attached to other registry objects (for example, principals).

Default ACLs

Object ACL
unauthenticated r----
user *creator* **rcidm**
other_obj r----
any_other r----

Created By
DCE configuration

The sec/group Directory

The following describes the namespace entries for /**.:/sec/group**, the security **sec/group** directory.

Name /**.:/sec/group/acct-admin**

Well Known
No

Description
This is the only group of principals that can create accounts.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID Generated

Created By
DCE configuration

Name /**.:/sec/group/bin**

Well Known
No

Description
This is the group for system binaries.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID 3

Created By
Security configuration

Name /**.:/sec/group/daemon**

Well Known
No

Description
This is the group for daemons.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID 1

Created By
Security configuration

Name /**.:/sec/group/kmem**

Well Known
No

Description
This is the group that has read access to kernel memory.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID 4

Created By
Security configuration

Name /**.:/sec/group/mail**

Well Known
No

Description
This is the group for the mail subsystem.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID 6

Created By
Security configuration

Name /**.:/sec/group/nogroup**

Well Known
Yes

Description
This is the default group for NFS access (user ID **nobody.**).

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID -2

Created By
Security configuration

Name /**.:/sec/group/none**

Well Known
Yes

Description
This member does not belong to a group. It is the default group.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID 12

Created By
Security configuration

Name /**.:/sec/group/subsys**

Well Known
Yes

Description
This directory contains **dce**. (See /**.:/subsys** in the CDS namespace.)

Default ACLs

Object ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----
Initial Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rcDnfmM
other_obj r-t-----
any_other r-t-----
Initial Container ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----

Created By
DCE configuration

Name /**.:/sec/group/system**
Well Known
No
Description
This is the group for system accounts.
Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID 0
Created By
Security configuration

Name /**.:/sec/group/tcb**
Well Known
No
Description
This is the group used by security policy daemons on OSF/1 C2/B1 secure systems.
Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID 18

Created By
Security configuration

Name /.:/sec/group/tty
Well Known
No
Description
This is the group that has write access to terminals.
Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----
UNIX GID 7
Created By
Security configuration

Name /.:/sec/group/uucp
Well Known
No
Description
This is the group for the UUCP subsystem.
Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----
UNIX GID 2
Created By
Security configuration

The sec/group/subsys Directory

The following describes the namespace entries for **/.:/sec/group/subsys**, the security **sec/group/subsys** directory.

Name /.:/sec/group/subsys/dce
Well Known
Yes
Description
This directory contains DCE required groups.
Default ACLs

Object ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn

other_obj r-----
any_other r-----

Initial Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rcitDnfmM
other_obj r-t-----
any_other r-t-----

Initial Container ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----

Created By
DCE configuration

Name /**.:/sec/group/subsys/dce/cds-admin**

Well Known
No

Description
This is the administrative group that is on the default ACLs for administrative objects. Clearinghouses have this group on their ACLs with all rights. The first user of the cell must be added to this group immediately after creation.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rcDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID Generated

Created By
DCE configuration

Name /**.:/sec/group/subsys/dce/cds-server**

Well Known
Yes

Description
This is the group of all CDS servers for the local cell. As each new server is added to the cell, it must be added to this group. CDS server authentication consists of checking for the server's membership in this group.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
group subsys/dce/cds-admin rctDnfmM
group subsys/dce/cds-server rctDnfmM
other_obj r-t-----

any_other r-t-----

UNIX GID Generated

Created By

DCE configuration

Name /**.:/sec/group/subsys/dce/dfs-admin**

Well Known

No

Description .

This is the DFS administrator's group. Members of this group have full permission to alter the DFS configuration within the cell.

Default ACLs

Object ACL

unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID Generated

Created By

DCE configuration

Name /**.:/sec/group/subsys/dce/dfs-bak-servers**

Well Known

Yes

Description

This is the security group to which all backup database servers belong. A server entry in the CDS group /**.:/subsys/dce/fs** is checked for authorization to act as a backup database server by determining whether it belongs to this security group.

Default ACLs

Object ACL

unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID Generated

Created By

DCE configuration

Name /**.:/sec/group/subsys/dce/dfs-fs-servers**

Well Known

Yes

Description

Abbreviated forms of the DFS server principals of all Fileset Database machines are in this group. The abbreviated form of a machine's DFS server principal stored in the group is of the form **hosts**/*dce_hostname*/**dfs-server**. A server entry obtained from the CDS group /**.:/fs** is checked for authorization to act as a fileset location server by determining if it belongs to this group.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
group subsys/dce/dfs-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID Generated

Created By
DCE configuration

Name **/.:/sec/subsys/dce/dts-admin**

Well Known
No

Description
This is the DTS administrator's group. Members of this group have full permissions to administer DTS by adding servers and so forth.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID Generated

Created By
DCE configuration

Name **/.:/sec/group/subsys/dce/dts-servers**

Well Known
Yes

Description
This is the group of DTS servers.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
group subsys/dce/dts-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID Generated

Created By
DCE configuration

Name **/.:/sec/group/subsys/dce/sec-admin**

Well Known
No

Description
This is the security administrator's group. Members of this group have full permissions to administer the security database.

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID Generated

Created By
DCE configuration

Name **/.:/sec/group/subsys/dce/audit-admin**

Well Known
No

Description
This is the audit daemon administrator's group. Members of this group have full permissions to administer the audit daemon (**auditd**).

Default ACLs

Object ACL
unauthenticated r-t-----
user *creator* **rctDnfmM**
group_obj r-t-----
group acct-admin rctDnfmM
other_obj r-t-----
any_other r-t-----

UNIX GID Generated

Created By
DCE configuration

The sec/principal Directory

The following describes the namespace entries for **/.:/sec/principal**, the security **sec/principal** directory.

Name **/.:/sec/principal/bin**

Well Known
No

Description
This is the owner of the system binaries.

Default ACLs

Object ACL
unauthenticated r--------
usr_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r------g
any_other r-------

UNIX UID 3

Created By
Security configuration

Name **/.:/sec/principal/cell_admin**
Well Known
No
Description
This is the principal who does the initial cell configuration.
Default ACLs

Object ACL
unauthenticated r-------
usr_obj rcDnfmaug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r------g
any_other r-------
UNIX UID Generated
Created By
Security configuration

Name **/.:/sec/principal/daemon**
Well Known
No
Description
This is the user for the various daemons.
Default ACLs

Object ACL
unauthenticated r-------
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r------g
any_other r-------
UNIX UID 1
Created By
Security configuration

Name **/.:/sec/principal/dce-ptgt**
Well Known
Yes
Description
This is the architecturally defined principal name of the privilege server.
Default ACLs

Object ACL
unauthenticated r-------
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------
UNIX UID 20

Created By
Security configuration

Name /.:/sec/principal/dce-rgy
Well Known
Yes
Description
This is the architecturally defined principal name of the registry server.
Default ACLs

Object ACL
unauthenticated r-------
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------

UNIX UID 21
Created By
Security configuration

Name /.:/sec/principal/hosts
Well Known
No
Description
This directory contains all DCE host principals.
Default ACLs

Object ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----
Initial Object ACL
unauthenticated r--------
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------
Initial Container ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----

Created By
Security configuration

Name /.:/sec/principal/krbtgt (also known as /...)
Well Known
Yes

Description

This is the architecturally specified name of the security namespace where foreign cell names are cataloged. All cells that this cell communicates with appear here.

Default ACLs

Object ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----

Initial Object ACL
unauthenticated r--------
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------

Initial Container ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----

Created By

Security configuration

Name /**.:/sec/principal/krbtgt/***cellname* (also known as /**.:**)

Well Known

No

Description

This is the principal of the authentication server of the cell named /**.../***cellname*. In the local cell, this is the principal for /**.:**.

Default ACLs

Object ACL
unauthenticated r-------g
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------

Created By

Security configuration

Name /**.:/sec/principal/mail**

Well Known

No

Description

This is the user for the mail subsystem.

Default ACLs

Object ACL
unauthenticated r--------
user_obj r---f--ug

user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------

UNIX UID 6

Created By

Security configuration

Name /.:/sec/principal/nobody

Well Known

No

Description

This is the default user for NFS access.

Default ACLs

Object ACL

unauthenticated r--------
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------

UNIX UID 2

Created By

Security configuration

Name /.:/sec/principal/root

Well Known

No

Description

This is the local operating system superuser.

Default ACLs

Object ACL

unauthenticated r--------
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------

UNIX UID 0

Created By

Security configuration

Name /.:/sec/principal/sys

Well Known

No

Description

This is a user who is permitted to read devices but is not a superuser.

Default ACLs

Object ACL

unauthenticated r--------
user_obj r---f--ug
user *creator* **rcDnfmaug**

group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------

UNIX UID 2

Created By

Security configuration

Name **/.:/sec/principal/tcb**

Well Known

No

Description

This is the user for security policy daemons on OSF/1 C2/B1 secure systems.

Default ACLs

Object ACL
unauthenticated r--------
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------

UNIX UID 9

Created By

Security configuration

Name **/.:/sec/principal/uucp**

Well Known

No

Description

This is the user for the UUCP subsystem.

Default ACLs

Object ACL
unauthenticated r--------
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------

UNIX UID 4

Created By

Security configuration

Name **/.:/sec/principal/who**

Well Known

No

Description

This is the user for remote **who** access.

Default ACLs

Object ACL
unauthenticated r--------
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug

other_obj r-------g
any_other r--------

UNIX UID 5

Created By

Security configuration

The sec/principal/hosts Directory

The following describes the namespace entries for /**.:/sec/principal/hosts**, the security **sec/principal/hosts** directory.

Name /**.:/sec/principal/hosts**/*dce_hostname*

Well Known

No

Description

This directory contains security principals for host *dce_hostname*.

Default ACLs

Object ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----

Initial Object ACL
unauthenticated r-------g
other_obj r---f--g
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
other_obj r-------g
any_other r--------

Initial Container ACL
unauthenticated r-----
user *creator* **rcidDn**
group acct-admin rcidDn
other_obj r-----
any_other r-----

Created By

Security configuration

Name /**.:/sec/principal/hosts**/*dce_hostname*/**cds-server**

Well Known

No

Description

The CDS server on node *dce_hostname* runs as this principal. This principal must be a member of the /**.:/subsys/dce/cds-server** group.

Default ACLs

Object ACL
unauthenticated r--------
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfma-g
group subsys/dce/cds-admin rcDnfma-g
other_obj r-------g

any_other r--------

UNIX UID Generated

Created By

Security configuration

Name /**.:/sec/principal/hosts**/*dce_hostname*/**dfs-server**

Well Known

No

Description

This is the principal name of the DFS servers on node *dce_hostname*.

Default ACLs

Object ACL

unauthenticated r-------g
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfma-g
other_obj r-------g
any_other r--------

UNIX UID Generated

Created By

Security configuration

Name /**.:/sec/principal/hosts**/*dce_hostname*/**gda**

Well Known

No

Description

The GDA on node *dce_hostname* runs as this principal. This principal must be a member of the /**.:/sec/group/subsys/dce/cds-servers** security group.

Default ACLs

Object ACL

unauthenticated r-------g
user_obj r---f--ug
user *creator* **rcDnfmaug**
group acct-admin rcDnfmaug
group subsys/dce/cds-admin rcDnfmaug
other_obj r-------g
any_other r--------

UNIX UID Generated

Created By

DCE configuration

Name /**.:/sec/principal/hosts**/*dce_hostname*/**self**

Well Known

Yes

Description

This entry is the principal for the DCE host *dce_hostname*. The security validation service of the **dced** daemon uses this principal. This is also the identity that local root processes can inherit.

Default ACLs

Object ACL

unauthenticated r--------
user_obj r---f--ug

user *creator* **rcDnfma-g**
group acct-admin rcDnfma-g
other_obj r-------g
any_other r--------

UNIX UID Generated

Created By
DCE configuration

Valid Characters and Naming Rules for CDS

Introduction — Valid Characters and Naming Rules

This section discusses the valid character sets for DCE Directory Service names as used by CDS interfaces. It also explains some characters that have special meaning and describes some restrictions and rules regarding case matching, syntax, and size limits. It is not a comprehensive reference for CDS, GDS, and DNS, but instead gives an overview of some key points to remember about each service. For specific information on valid characters in GDS and DNS names, see the documentation for each technology.

More Detailed Information — Valid Characters and Naming Rules

The use of names in the DCE often involves more than one directory service. For example, CDS interacts with either GDS or DNS to find names outside the local cell.

Note: Because CDS, GDS, and DNS all have their own valid character sets and syntax rules, the best way to avoid problems is to keep the names short and simple, consisting of a minimal set of characters common to all three services. The recommended set is the letters A to Z, a to z, and the digits 0 to 9. In addition to making directory service interoperations easier, use of this subset decreases the probability that users in a heterogeneous hardware and software environment will encounter problems creating and using names.

Figure A-8 on page A-49 details the valid characters in CDS names and the valid characters in GDS and DNS names as used by CDS interfaces:

- Characters in white boxes are valid in all three kinds of names.
- Characters in light shaded boxes are valid only in CDS and GDS names.
- Characters in dark shaded boxes are valid only in CDS names.

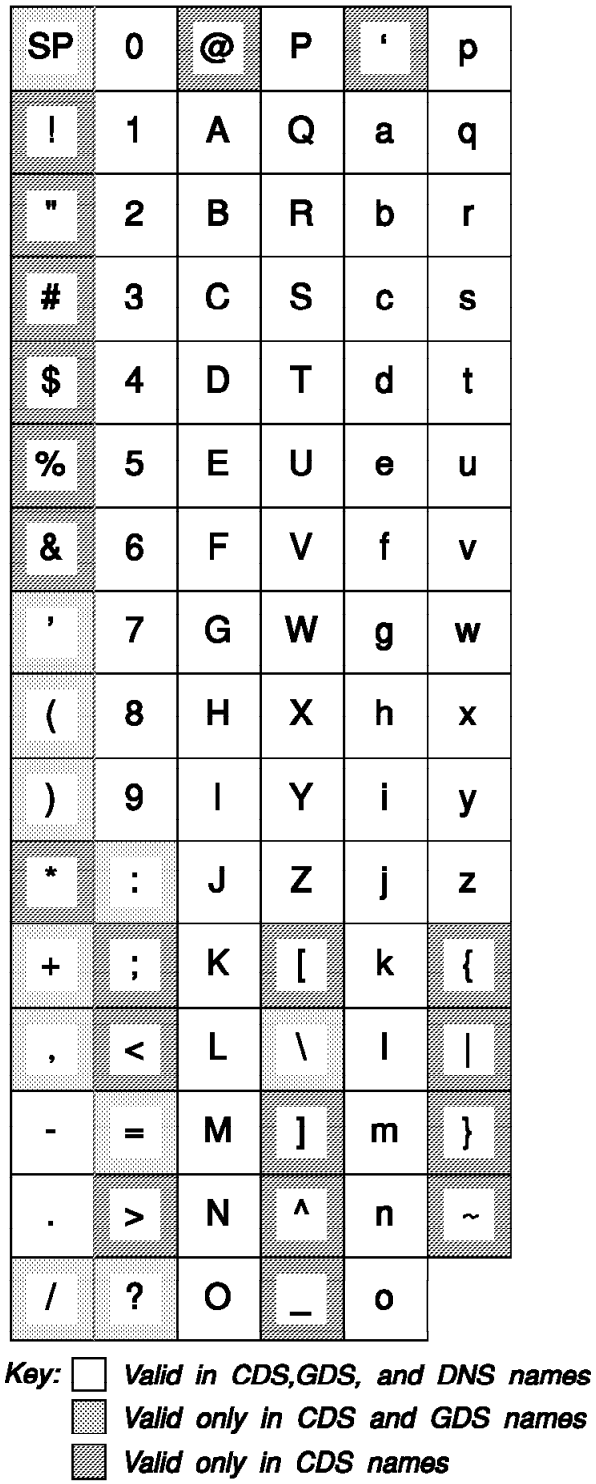


Figure A-8. Valid Characters in CDS, GDS, and DNS Names

Notes:

1. The slash (/) and backslash (\) characters can only be used as metacharacters and not as actual characters in a name. See “Metacharacters” on page A-5 for more information.
2. The ! and # symbols are comment characters. When used in CDSCP commands, these characters and everything after them are treated as comments and ignored.
3. Security registry names generally contain only characters from the Portable Character Set (PCS), excluding the @ and : characters. However, you can also use an extended character set as described in “Principal, Group, and Organization Names” on page 7-106.

Although spaces are valid in both CDS and GDS names, a CDS simple name containing a space must be enclosed in quotes (“”) when you enter it through the CDS control program. Additional interface-specific rules are documented in the modules where they apply.

For additional information, see:

- “Metacharacters” on page A-5
- “Additional Rules” on page A-1
- “Maximum Name Sizes” on page A-5

World Time Zone Map

Figure A-9 on page A-51 shows a map of the world time zones, including the following:

- UTC reference zone
- Odd- and even-numbered zones
- Half-hour zones
- Countries and areas that have not adopted the zone system or where time differs other than a half hour from the neighboring zone

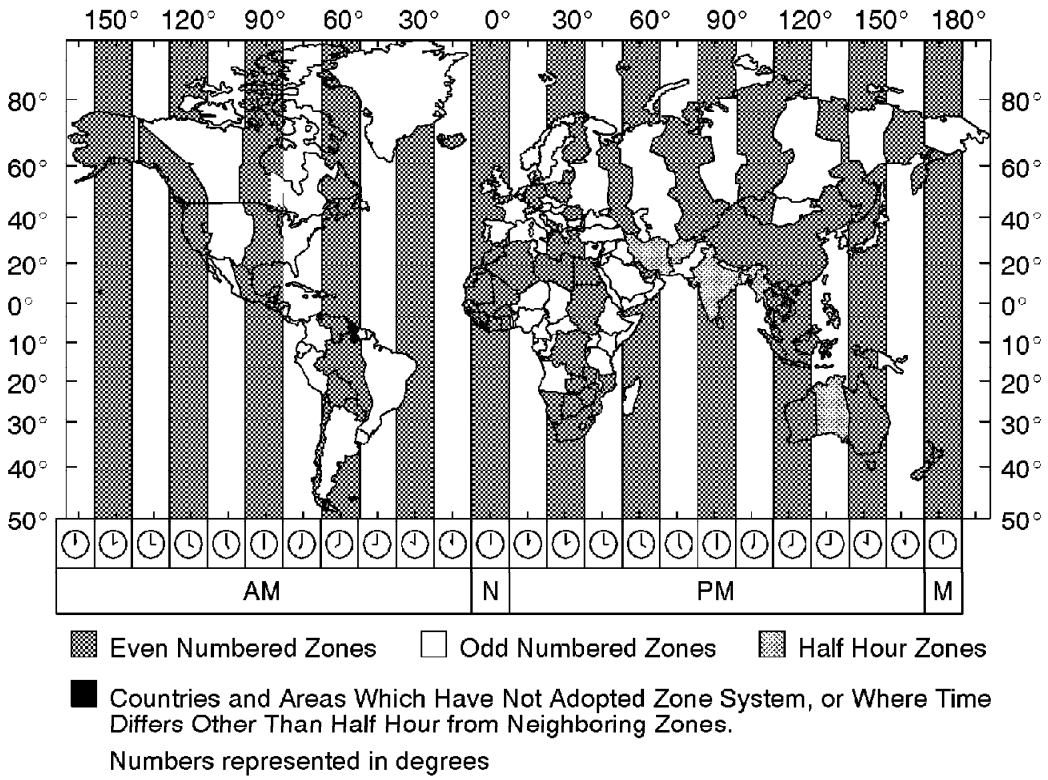


Figure A-9. World Time Zone Map

Index

Special Characters

_b(sec), last security server used 2-8
_c, current cell name 2-5
_h, current host name 2-6
_n, most recent operation argument name 2-6
_n(_p), parent of 2-6
_o, last dcecp object name 2-6
_r, last operation's return value 2-6
_s(sec) variable 7-138
_s(xxx), DCE servers to use 2-7
_u (user), current principal name 2-5
/: prefix 5-13
/... prefix 5-12
/. prefix 5-12
/etc/group file, login 7-141
/etc/passwd file, login 7-141
/var/dce A-4
“” (quotation marks) 2-37
{} (braces) 2-37
\ (backslashes) 2-37

A

Abbreviations 2-16
absolute time 1-40
access control
  in the namespace 1-20
  planning 1-11
Access Control Lists (ACL)
  checking sequence 7-7
  copying to other objects 7-45
  default 7-39
  denying access 7-8
  displaying 7-68
  entries 7-1
  entry types for principals and groups 7-1
  function 1-45
  introduction 1-3
  masks 7-1
  scope 1-46
  scope compared to AIX permission bits 1-46
  syntax 7-1
  types of, editing 7-39
  using 1-59
access control, setting up 5-62
access method identification 7-28
access to CDS names, controlling 1-29
access to namespace by CDS servers 5-10
accounts
  adding 7-106
  authentication 7-85
  automatic changes to locksmith 7-35
accounts *(continued)*
  changing 7-13
  created by registry connect command 7-61
  creating 7-9
  creating and maintaining 1-48
  defined 7-142
  deleting 7-13
  effects of creation on membership lists 7-91
  expiration information 7-87
  lifespan 7-104
  machine 7-90
  machine accounts 7-90
  managing 1-21
  modifying cross-cell authentication 7-94
  reserved 7-129
  server accounts 7-137
  server machine 7-92
  steps to add 7-13
  summary of information 7-10
  UNIX, importing to DCE 1-21
  user 7-146
ACL entries, checking 5-16
ACL entry types
  compared to ACL types 7-39
  for principals 5-1
  in future DCE releases 7-5
ACL interpretation 1-47
ACL management tasks 7-9
ACL Manager
  checking sequence 7-5
  role in granting access 1-46
  support for entry types 7-5
adding
  accounts 7-9
  comments to scripts 2-23
  group members 7-91
  groups 7-14
  members to a group entry in CDS 4-27
  members to groups 7-134
  members to namespace authorization group 5-62
  new attributes A-7
  new objects 2-13
  organizations 7-14
  principals 7-17
  profile members 4-29
  scripts to dcecp sessions 2-13
  steps 7-13
additional file systems to create A-4
address changes, simultaneous 7-85
adjtime() limitations 1-42
adjusting system clocks 1-39

administering a multicell environment 1-45
administration
client 4-1
DCE applications 1-27
easing applications 4-15
simplifying 3-22
task objects 1-26
task objects, using 3-22
administration objects 2-15
administration programs
CDS 1-17
cdsbrowser 1-17
cdscp 1-17
DCE Audit 7-19
DFS 1-17
DTS 1-17
passwd_export 1-17
passwd_import 1-17
passwd_override 1-17
rpccp 1-17
sec_admin 1-17
sec_create_db 1-17
administration tools 1-18
administration utilities 1-16
administrative tasks, DCE Audit Service 1-49
adopting registry orphans 7-23
advertise command, dtscp 6-10
advertising global servers 6-19
AIX/DCE security integration 7-24
aliases 7-107
American National Standards Institute 1-7
ANSI 1-7
answering prompts 7-103
append procedures, overview 5-55
application development environment machines 1-18
applications
development 4-15
applications support 6-5
assigning courier roles 6-20
attribute instances
attaching to objects 7-47
creating 7-46
deleting 7-48
maintaining 7-46
modifying 7-47
attribute schema definition 1-49
attribute sets, using 7-48
attribute types
access control 7-143
creating 7-49
deleting 7-51
modifying 7-51
renaming 7-51
attributes
checkinterval 6-20
globaltimeout 6-25
attributes *(continued)*
localtimeout 6-25
maxinaccuracy 6-23
minservers 6-22
queryattempts 6-25
serverentry 6-26
tolerance 6-24
audit 1-49
Audit clients definition 1-50
Audit code point 1-50
Audit daemon
ACL permissions 7-43
attributes 7-94
controlling access to 7-43
controlling audit trails 7-44
giving permissions to 7-43
initial ACL 7-43
starting 7-140
Audit event 1-51
Audit event class
file 1-51
file, SEP line 7-66
grouping 1-51
how to define 7-65
name 1-51
number 1-52
number format 1-52
Audit event number 1-51
Audit filter
audit actions 1-54
audit condition 1-54
creating and maintaining 7-54
guides 1-54
key 1-54
logging audit records 1-53
subject identity 1-53
types 1-53
Audit filter rules
high-water-mark rule 1-54
override rule 1-54
Audit filters, enabling 7-55
audit logging, enabling and disabling 7-74
Audit server processes 1-15
Audit Service
administration tasks 7-20
application programmer's tasks 7-19
client programs 1-13
components 1-50
concepts 1-50
features 1-50
Audit trail file
changing storage option 7-45
controlling 7-44
defined 1-54
displaying 7-44

audit trail file storage option, modifying 7-45
audit-admin group 1-12
authentication
  clients 1-56
  Extended Security Attributes 7-80
  master and slave replicas 7-129
  policy 7-33
  service servers 1-56
Authentication Service
  how it works 7-9
  security overview 1-44
  shared authentication keys 7-144
authorization groups
  adding members 5-62
  creating 5-62
authorization overview 1-45
automatic changes to locksmith account 7-35

B

backslashes (\) 2-37
backup procedures
  CDS 3-1
  registry database 7-36
  Security Service Registry 3-1
backup procedures, namespace 5-5
Backus Naur Format (BNF) for DTS A-2
balancing server workloads 4-25
BIND namespace 5-4
binding information
  exporting 4-24
  importing from a group entry 4-27
  importing from a profile entry 4-30
  importing from a server entry 4-24
  removing 4-24
braces ({}) 2-37
break terminate loops 2-2
browser 1-18
  cdsbrowser utility 1-18
  displaying the namespace 5-28
  how to use 5-37

C

Cache Manager 1-14
caching introduction 1-4
case command 2-2
case-matching rules A-1
CDS
  Access Control List (ACL) 5-54
  administration programs 1-17
  appending directories 5-47
  attributes file A-9
  backup procedures 5-5
  case-matching rules A-1
  cdscp commands 5-10
CDS *(continued)*
  clearinghouse 1-18
  component of the naming environment 5-26
  components 5-8
  control program 1-18
  control programs for managing 1-31
  CTS 5-37
  dcecp commands for managing 5-42
  deleting non-replicated directories 5-22
  deleting replicas 5-22
  displaying attribute values 5-38
  elements ordering A-1
  internal data storage A-1
  listing contents of directories 5-37
  local management operations access 5-14
  lookups 1-30
  maintenance tasks 1-18
  managing 1-19
  managing configuration 1-30
  merging directories 5-46
  metacharacters A-5
  monitoring 1-19
  name sizes, maximum A-5
  names 5-3
  object attributes 5-43
  object types 5-42
  operation 1-28
  permissions 5-20
  restoring merged directories 5-60
  sample configuration (figure) 5-8
  security 5-61
  updating data 1-34
  user interface 5-11
  UTS 5-37
CDS attribute
  adding A-7
  modifying A-7
  removing A-7
CDS clerk
  clearinghouse communications 5-53
  client programs 1-13
  defined 5-8
  disabling 5-28
  viewing counters 5-54
CDS directory
  hosts A-18
  subsys A-23
  top level A-13
CDS names, controlling access 1-29
CDS server clerks, synchronizing 5-64
CDS Servers
  definition 5-8
  node components (figure) 1-33
  permissions 5-10
  security 5-61
  system upgrade 5-56

CDS space A-12
cds-admin group 1-12
cds_attributes file A-9
cds_attributes file, modifying A-8
cds_clerk.h file A-6
CDS_GDAPointers attribute 5-35
cds_globalnames file A-11
cds_globalnames file, modifying A-8
cdsadv 1-13
cdsadv command 5-46
cdsbrowser 1-17
cdsclerk 1-13
cdscp
  commands 5-10
  description 5-10
  managing the namespace 1-17
  show command 5-38
cdsd 1-15
cell entry
  creating in DNS 5-31
  creating in GDS 5-32
cell names
  conventions 1-6
  current (_c) 2-5
  establishing 1-6
cell namespace
  boundaries 1-8
  CDS Browser 1-18
  collapsing 5-30
  entry types 1-9
  expanding 5-30
  replication 1-11
  security 1-20
cell objects, modifying 3-13
cell operation, testing 3-21
cell stability 1-9
cell-relative names 5-12
cell, introduction 1-2
cells
  communication between 1-5
  entry creation 5-31
  naming environment 5-25
  planning 1-4
  WAN 6-8
change command, dtscp 6-1
changes to locksmith account, automatic 7-35
changing aliases to primary names 7-46
changing network address of a DCE machine 1-18
changing primary names to aliases 7-46
character sets A-1
characters, valid A-5
check mode 7-102
checking ACL entries 5-16
checking sequence
  ACLs 7-5
  permissions 7-5
checkinterval attribute 6-20
child pointers
  description 1-33
  example 5-66
clearinghouse create command 5-58
clearinghouse delete command 5-21
clearinghouse disable command 5-58
clearinghouses
  clerk communications 5-54
  deleting 5-21
  description 5-9
  disabling 5-28
  displaying contents 5-28
  displaying counters 5-54
  forcing to checkpoint to disk 5-34
  monitoring clerk communications 5-53
  monitoring counters 5-54
  object entries 1-32
  preserving across system upgrade 5-56
clerk management operations access 5-14
clerks
  creating 6-30
  disabling 5-28
  DTS 6-9
  modifying attributes 6-21
  monitoring counters 5-54
  restarting 5-59
  stopping 6-30
client administration 4-1
client considerations 1-12
client machine requirements 1-12
client programs
  Audit service 1-13
  CDS 1-13
  DFS 1-14
  DTS 1-13
  RPC 1-13
client/server binding information, managing 4-17
clients and servers, introduction 1-1
clients, authentication 1-56
clock error 1-36
code point 1-50
collapsing cell namespace 5-30
collapsing selected directories 5-29
command language overview 1-23
command lines, editing 2-23
command operations, DTS 6-17
command substitution usage 2-1
command syntax, passwd_import 7-141
components of DCE Audit Service 1-50
configuration
  common server needs 4-3
  DTS for WANs and WAN links 6-6
configure command, dtscp 6-4
configured DCE servers and hosts, displaying 3-10

configuring a DCE host 3-3
configuring DTS for a LAN 6-3
constructing strings 2-42
containers
  containers.definition of 7-39
  default ACLs 7-40
continue terminate loops 2-2
control program
  dcecp command usage 2-19
  introduction 1-22
  language usage 1-24
  managing the Security Service 1-47
control program usage 2-15
control program, DCE 1-17
controlling access
  Audit daemon 7-43
  DTS 6-4
  management operations 5-14
controlling access to CDS names 1-29
controlling audit trails 7-44
controlling interpretation 2-37
controlling scripts 2-1
controlling server operation 4-3
convenience variables 2-5
Coordinated Universal Time (UTC)
  See UTC (Coordinated Universal Time)
copying
  ACLs 7-45
  clearinghouse database files 5-58
counters, DTS 6-18
counting with expressions 2-44
courier role 6-10
courier role, assigning 6-20
courier servers, designing 6-19
create directory command 5-17
create link command 5-43
create replica command 5-18
create, file systems A-4
creating
  accounts 1-48
  additional authorization groups 5-62
  attribute instances 7-46
  directories 5-15
  filters 7-54
  groups 1-48
  keytab files 3-5
  new clerk or server 6-30
  new commands with proc 2-11
  new group entries in CDS 4-26
  new profiles 4-29
  new users 3-3
  organizations 1-48
  override file entries 7-97
  principals 1-48
  read-only replicas 5-17
  server entries in CDS 4-24
creating *(continued)*
  slave replicas 7-56
  soft links 5-43
  trust relationships 7-60
  with eval 2-10
creating aliases 7-46
creating and maintaing aliases
  groups 7-46
  principals 7-46
creating attribute types 7-49
creating the primary registry database 7-56
Creation Time Stamp (CTS) 5-37
credential inherited by processes 1-47
credentials 1-47
CTS
  See Creation Time Stamp (CTS)
current cell name (_c) 2-5
current host name (_h) 2-6
current principal (user) name (_u) 2-5
customizing dcecp sessions 2-13

D

daemons
  cdsd 1-15
  dced 1-13
  dfsd 1-14
  dtsd 1-13
  gdad 1-15
  secd 1-14
data consistency 5-37
database, determining most current 7-111
DBCS 2-16, 3-5
  deleting 3-5
  maintaining 3-5
DCE administration objects 2-15
DCE administration utilities 1-16
DCE Audit
  administration 7-19
  programming 7-19
DCE Audit environment variables, setting 7-138
DCE control program
  commands for managing CDS 5-41
  creating audit filters 7-54
  description 5-41
  enabling and disabling audit logging 7-74
  help facilities 2-21
  modifying and querying Audit daemon attributes 7-94
  modifying filters 7-54
  Security Service objects 2-16
  starting 2-16
  stopping 2-16
DCE control program usage 2-15
DCE daemon, starting and stopping 4-13

DCE host, removing from cell 3-14
DCE hosts
  configuring 3-3
  listing 3-12
  managing 1-26
DCE maintenance overview 1-18
DCE processes, stopping 3-20
DCE Security Service, validating authenticity 7-146
DCE server machine files 1-14
DCE servers to use (_s(xxx)) 2-7
DCE users, managing 1-27
dcecp 2-15
dcecp account create command 7-138
dcecp commands
  account catalog 7-67
  account create 7-13
  account delete 7-13
  group create -uuid 7-23
  group list 7-70
  keytab add 3-6
  keytab delete 3-8
  keytab remove 3-8
  list 5-37
  modify A-7
  org create -uuid 7-23
  principal catalog 7-72
  principal create -uuid 7-23
  properties 7-110
  registry modify 7-35
  registry show 7-35
  remove A-7
  show 5-38
  usage 2-19
dcecp keytab add command 7-138
dcecp keytab remove command 7-138
dcecp log object 4-35
dcecp objects, writing 1-25
dcecp program usage 5-41
dcecp security commands, using names 7-136
dcecp, customizing sessions 2-13
dced 1-13
  rpccp 1-13
default filters 7-55
defining
  attribute trigger servers 7-63
  event classes 7-65
defining ACL managers for attributes 7-51
defining attribute type encoding 7-52
delete replica command 5-25
deleting
  accounts 7-13
  aliases 7-18
  attribute instances 7-48
  attributes A-7
  clearinghouses 5-21
  DCE host from cell 3-14
deleting *(continued)*
  directories 5-22
  directory replicas 5-22
  expired entries in local registry 7-93
  filters 7-54
  group entry from CDS 4-27
  group members 7-91
  groups 7-16
  keytab files 3-5
  non-replicated directories 5-22
  organizations 7-16
  principals 7-18
  profile entries 4-30
  read-only replicas 5-24
  server entries in CDS 4-24
  servers from network 7-125
  slave replicas 7-83
  soft links 5-44
  users 3-9
deleting attribute types 7-51
designating a new master replica 5-51
designing courier servers 6-19
designing global servers 6-19
determining entry names 4-1
determining most current database 7-111
determining naming environments 5-12
DFS administration programs 1-17
DFS client cache A-4
DFS processes, dfsd 1-14
dfs-admin group 1-12
dfsbind 1-14
dfsd 1-14
direct trust relationships 7-144
directories
  appending errors 5-33
  child 5-3
  child pointers in CDS 1-33
  collapsing 5-29
  controlling access to 1-59
  convergence values defined 5-50
  creating 5-15
  deleting 5-22
  expanding 5-29
  managing in CDS 1-30
  merging errors 5-33
  merging in CDS namespace 1-34
  merging into foreign cells 5-49
  modifying convergence 5-50
  modifying replica set 5-51
  parent 5-3
  restructuring 1-34
  skulk operation 5-64
directory create command, entering 5-16
directory delete command 5-23
directory modify command 5-50

Directory Services, DCE component usage 5-25
directory show command 5-51
directory synchronize command 5-64
directory version
  upgrading on a directory 5-16
  upgrading on cell root 5-16
disable clerk command 5-28
disable server command 5-28
disabling
  audit logging 7-74
  servers 4-5
disassociating a clearinghouse 5-58
displaying
  account information 7-67
  ACL information 7-68
  all servers 3-11
  audit trail files 7-44
  CDS name attribute values 5-38
  clearinghouse counters 5-54
  clerk attribute information 5-38
  clerk counters 5-54
  configured DCE servers and hosts 3-10
  default namespace 5-29
  group information 7-69
  information in endpoint map 4-16
  keytab information 7-71
  namespace content 1-30
  namespace structure 1-30
  namespace with CDS Browser 5-28
  organization information 7-69
  principal information 7-72
  privilege attributes and tickets 7-86
  registry information 1-57
  server attribute information 5-38
  server counters 5-54
  user information 3-11
Distributed Computing Environment (DCE)
  introduction 1-1
DNS 1-8
documenting scripts 2-23
Domain Name System 1-8
Domain Name System (DNS) cell names 1-7
drift 1-37
DTS
  absolute time 1-40
  access control list (ACL), using 6-4
  adjtime(), using 1-42
  administration programs 1-17
  applications support 6-4
  changing system time 6-1
  clerks 6-9
  clock error 1-36
  communications delays 1-36
  components 6-9
  configuration planning 6-3
  configuring for a LAN 6-3
DTS *(continued)*
  courier role 6-10
  entity, creating 6-30
  features and benefits 1-35
  inaccuracy measurement 6-6
  inaccuracy values 1-36
  introduction 1-34
  managing 1-43
  planning 1-16
  planning guidelines 6-12
  planning implementation 1-43
  planning team 1-43
  reconfiguring on nodes 6-30
  relative time 1-41
  server management 6-19
  server programs 1-16
  servers 6-9
  syntax in extended BNF notation A-2
  system clocks, synchronizing 1-37
  temporarily reconfiguring 6-31
  time measurement factors 1-36
  Time Provider Interface (TPI) 6-5
  time providers 6-5
  time representation 1-40
  timestamp syntax 6-12
  usage 1-36
  UTC 1-35
DTS command operations 6-17
DTS counters 6-18
DTS maintenanc tasks 1-20
DTS object attributes 6-18
DTS objects 6-17
dts-admin group 1-12
dts_ntp_provider.c program 6-13
dts_null_provider.c program 6-13
dtscp 1-17
dtsd 1-13
duplicate names 5-33

E

editing ACLs on CDS names 5-31
editing ACLs, mask effect 7-40
editing command lines 2-23
effects of account creation 7-91
elements ordering A-1
elements, grouping 2-37
enabling
  audit logging 7-74
  servers 4-5
enabling Audit filters 7-55
enabling DCE access 7-24
endpoint maps
  about 4-15
  displaying information 4-16

endpoints
purging obsolete 4-15
restricting 4-16
entries, ACL 7-1
entries, cell namespace 1-9
entries, object 1-32
entry name, determining 4-1
entry names, providing to clients 4-2
entry types, cell namespace 1-9
entry types 1-10
environment machines, application development 1-18
environment variables
setting DCE Audit 7-138
usage 4-37
epoch number 6-20
error handling
appending and merging 5-33
traps 2-27
error information variables, global 2-27
errors, reissuing 2-28
establishing a cell name 1-6
eval command 2-10
event class
defining 7-65
file 1-51
grouping 1-51
name 1-51
number 1-52
event number 1-51
examing group entries 7-132
examining password file 7-133
exception handling, traps 2-27
excluding replica from set 5-51
expanding cell namespace 5-30
expanding selected directories 5-29
expired credentials files, removing 1-22
exporting binding information 4-24
expressions to measure and count 2-44
extended ACL entry type form and function 7-5
Extended Registry Attribute 1-49
Extended Security Attributes for principals 7-79
extending cell objects 3-13
extending server configurations 4-6

F

features, DCE Audit Service 1-50
file names, local 5-13
file, specifying names 2-49
Changing the IP Address 3-2
IP Address, changing 3-2
files
/etc/group 7-141
/etc/passwd 7-141
controlling access to 1-59
created at runtime A-4
files *(continued)*
reading 2-50
to create after installation A-4
writing 2-50
files installed on DCE server machines 1-14
files, working with 2-49
filespace introduction 1-3
filter guides 1-54
filtering the namespace display 5-30
filters, default 7-55
forcing system synchronization 6-2
forcing the clearinghouse to checkpoint 5-34
formal task objects 2-29
full name 5-3

G

gdad command 5-46
GDS 1-6
getting help from Registry Editor 7-113
Global Directory Agent (GDA)
operations 5-35
planning 1-15
processes 1-15
Global Directory Service (GDS) 1-6
global error information variables 2-27
global names 5-12
global planning 1-4
global servers
advertising 6-19
component of DTS 6-10
designing 6-19
use of minservers 6-23
globaltimeout attribute 6-25
group ACL entry types 7-1
group entries 4-25
group entries, AIX 7-134
group entry members, viewing 4-27
group IDs, setting in registry 7-108
group permissions 7-4
group_override file 7-97
grouping elements 2-37
groups
ACL entry types 7-1
adding members 7-91
created during configuration 1-12
deleting 7-16
deleting members 7-91
displaying registry information 7-69
entry, examining 7-132
excluding from project lists 7-4
modifying 7-15
name 7-106
overriding memberships 7-99

H

handling network address changes 7-84
help facilities, dcecp 2-21
history command 2-25
home directory, changing 7-99
host cell name information, modifying 4-20
host daemon directory A-21
host data, managing 4-19
host file data, manipulating 4-20
host name, current (_h) 2-6
host object, modifying or extending 3-13
host running test 3-22
host services, managing 1-28
host services, starting and stopping 4-13
host, removing from cell 3-14
host, starting configured DCE processes 3-20
hosts, configuring 3-3
hosts, managing 1-26

I

identical user option 7-102
idl 1-18
if statements 2-2
importing AIX accounts 1-57
importing binding information 4-22
inaccuracy values 1-36
informal administration scripts 2-39
Initial Container ACL 7-39
Initial Object ACL 7-39
interactive commands 2-40
intercell communication 1-5
intercell naming 1-31
interim file 5-55
internal data storage A-1
internationalization 5-15
interoperation with NTP 1-42
introduction to DCE 1-1
invalid login handling 7-82
invoking passwd_import 7-132

J

junctions, namespace 5-4

K

kdestroy command 7-88
key entries, deleting from keytab file 3-8
key tables, displaying registry information 7-71
keys
  deleting from keytab file 3-6
  machine and server accounts 3-5
  version numbers 3-6
keytab files
  adding keys 3-6
keytab files *(continued)*
  creating 3-5
  dced object 3-5
  modifying passwords 3-8
  protecting 3-6
kinit command 7-86
klist command 7-86, 7-87
krbgt directory 1-45

L

LANs connected by WAN links 6-7
LANs with WAN links 6-7
last dcecp object name (_o) 2-6
last operation's return value (_r) 2-6
last security server used (_b(sec)) 2-8
leap seconds 1-37
lifespan
  account 7-104
  password 7-105
link delete command 5-44
listing DCE hosts 3-12
listing server configuration information 4-5
lists, operating on 2-45
local file names 5-13
local files, ensuring consistency 7-74
local NTP time sources 6-13
local registry
  defined 7-141
  files 7-92
  purging expired entries 7-93
  setting capacity 7-93
  setting lifespan 7-93
  setting properties 7-93
local servers 6-10
localtimeout attribute 6-25
locating names 5-40
locksmith mode
  restarting Primary Server 7-131
  restarting Security server 7-130
  starting Security Server 7-131
login shell, changing 7-99
login, invalid 7-82
looing beyond tools 2-41
lookups, CDS 1-30
loops in script execution 2-2
loops, preventing 6-28

M

maintaining
  accounts 1-48
  attribute instances 7-46
  filters 7-54
  groups 7-14
  keytab files 3-5

maintaining *(continued)*
local registries 7-92
membership lists 7-91
organizations 7-14
policies 1-58
principals 7-17
properties 1-58
maintenance tasks
DTS 1-20
Security Service 1-21
maintenance, local registry 7-114
management tasks, ACL 7-9
managing
CDS 1-19
CDS directories 1-30
Cell Directory Services 1-31
cells 1-26
clearinghouses 1-30
clerks 1-30
client/server binding information 4-17
DCE hosts 3-3
DTS 1-43
Global Directory Agent (GDA) 5-45
host data 1-28
host services 1-28
hosts 1-26
intercell naming 1-31
Security Service 1-21
servers 1-30
soft links 5-43
users 1-27
managing host data 4-19
managing server entities 4-22
managing servers
entries 4-22
groups 4-22
profiles in CDS 4-22
manipulating data in host files 4-20
manipulating strings 2-42
map, world time zones A-50
mask effect when editing ACLs 7-40
maskobj entry type, effect on ACL checking 7-7
masks
ACL 7-1
cell entry creation 5-32
types and usage 7-4
master and slave replicas 7-126
master key, changing 7-38
master keys, restoring 7-37
masters, restoring duplicate 7-131
matching server epochs 6-20
maximum name sizes A-5
maximum permissions, establishing 5-63
maximum ticket renewable lifetime 7-33
maxinaccuracy attribute 6-23
measuring with expressions 2-44
membership lists, maintaining 7-91
merge procedures, overview 5-55
merging CDS directories overview 1-34
messages
routing serviceability 4-33
serviceability severity levels 4-34
metacharacters A-5
minimum ticket lifetime, setting in registry 7-109
minservers attribute 6-22
modifying
accounts 7-12
attribute instances 7-47
Audit daemon attributes 7-94
audit trail file storage option 7-45
CDS entity attributes A-6
cds_attributes file A-8
cds_globalnames file A-8
cell objects 3-13
clerk attributes 6-21
cross-cell authentication accounts 7-94
directory convergence 5-50
directory replica set 5-51
existing attribute values A-7
filters 7-54
groups 7-15
host object 3-13
organizations 7-15
passwords in keytab file 3-8
primary registry site 7-37
principals 7-17
server attributes 6-21
server configurations 4-6
soft link destination name 5-44
soft link expiration value 5-44
system time 1-20
user object 3-13
modifying attribute types 7-51
modifying host cell name information 4-20
monitoring
CDS 1-19
clerk communications 5-53
counters 5-54
monitoring program performance 4-33
most recent operation argument name (_n) 2-6
multicell environment, administering 1-45

N

name sizes, maximum A-5
names
aliases 7-107
cell 1-8
duplicate 5-33
format 7-107
full 7-107

names *(continued)*
locating 5-40
primary 7-107
security objects 7-135
unreachable 5-34
names to resources translation 5-65
namespace
cell 1-11
definition 1-10
introduction 1-3
navigating 5-30
planning 1-4
Security 1-10
namespace display, filtering 5-30
naming environment
determining 5-12
global names 5-12
relationship to cell 5-25
naming rules A-48
naming, intercell 1-31
navigating the namespace 5-30
network address changes 1-18
network address changes, handling 7-84
network reconfigurations 1-57
Network Time Protocol (NTP)
description A-9
interoperation 1-42
local time sources 6-13
provider programs 6-13
new user creation 3-3
non-PCS 5-15, 7-108
NTP
See Network Time Protocol (NTP)
ntp.conf file 6-15
ntpd daemon 6-15

O

Object ACL 7-39
object attributes, DTS 6-18
object creation quotas 7-95
object entries 1-32
object identifier
obtaining A-6
origin A-8
object identifier files
editing for unique identifiers A-8
translating 1-59
objects
controlling access to 1-59
quotas for creating 7-61
types of 7-39
objects and containers 7-39
objects, dcecp log 4-35
objects, DTS 6-17
OID A-6
OMIT entry 7-99
omitting users from password files 7-99
operating on lists 2-45
operating system
backups 5-5
commands run from scripts 2-47
operation
DTS 6-9
override 7-95
organization IDs, setting in registry 7-109
organizations
adding and maintaining 7-14
defined 7-142
deleting 7-16
displaying registry information 7-69
modifying 7-15
name 7-106
origin of object identifiers A-8
orphans, adopting 7-23
override entries
changing the home directory and login shell 7-99
creating 7-95
multiple entries 7-100
omitting users 7-99
passwords for local machines 7-98
preventing login 7-99
override operation 7-95
overrides, applying to all group members 7-99
overview
CDS 1-28
command language 1-23
DCE authorization for CDS 5-54
DCE maintenance 1-18
DTS 1-36

P

parent of _n(_p) 2-6
parseargs procedure 2-35
parsing strings 2-42
passwd_export 1-17
passwd_import
invoking 7-132
sample session 7-131
passwd_import command 1-57
administration programs 1-17
passwd_import command syntax 7-141
passwd_import generated registry entries 7-101
passwd_import operation 7-100
passwd_import usage 7-102
passwd_override 1-17
passwd_override file 7-96
password lifespan 7-105
password synchronization schema definitions 3-15

passwords
AIX group 7-134
effects of policy changes 7-34
encrypting 7-108
expiration 7-83
expiration date 7-105
expiration information 7-87
file entries 7-134
format 7-105
generation 3-17
hidden property 7-108
password file, examining 7-133
server account 7-138
specifying for a particular machine 7-98
strength 3-17
pe_site file, updating 7-84
permissions
accessing host data 4-19
accrual, effect on ACL checking 7-7
accruing 7-4
ACL entry syntax 7-1
Audit Service 7-43
CDS 5-20
cdscp commands 5-6
denying 7-8
giving to Audit clerks and administrators 7-43
granting, effect on ACL checking sequence 7-8
group 7-4
host data container 4-19
host data items 4-19
inheriting 1-47
principals and groups 7-4
project lists 7-4
propagating to CDS directories 5-57
registry 7-114
restricting 7-4
scope of AIX permission bits 1-46
setting default 7-39
setuid bit 1-47
supported by CDS 5-20
use of 1-45
planning
DTS implementation 1-43
External Time Providers 6-11
planning guidelines for DTS 6-12
planning sites for security service 7-103
point-to-point lines, use of minservers attribute 6-23
policies and properties definition 7-142
policy
handling conflicting 7-34
overrides 1-21
setting 7-104
setting and maintaining 1-21
standard 7-104
populating new registry databases 7-106
POSIX scope of permissions and DCE ACLs 1-46
scope of support 1-46
preventing login 7-99
preventing loops 6-28
principal ACL entry types 7-1
principal, current (user) name (_u) 2-5
principals
ACL entry types 5-1
adding 7-17
authenticating 7-9
displaying registry information 7-72
Extended Security Attributes 7-79
introduction 1-3
maintaining 7-17
modifying 7-17
name 7-106
overriding group memberships 7-99
reserved 7-129
unauthenticated 5-63
unauthenticated, maximum permissions 5-63
privilege attributes
obtaining 7-86
viewing 7-86
Privilege Service 1-44
proc 2-11
profiles 4-27
profiles, CDS namespace 1-10
program performance, monitoring 4-33
program usage
program usage, dcecp 5-41
programs, remote administration machines 1-17
prohibiting inclusion on project lists 7-15
project list permissions 7-4
project lists
excluding groups from 7-4
in ACL checking sequence 7-6
prohibiting inclusion 7-15
propagating database changes 7-129
propagating updates in CDS 5-69
propagation to directories and contents 5-57
protecting local resources
/etc/security/user file 7-26
group_override file 7-26
local system administrator 7-25
passwd_override file 7-26
providing entry names to clients 4-2
purging expired entries in local registry 7-93

Q

queryattempts attribute 6-25
querying
Audit daemon attributes 7-94
servers 4-7
questions for planning 1-5

quotas for creating registry objects 7-61
quotation marks (“”) 2-37

R

radio transmissions, UTC A-9
read-only replicas, deleting 5-24
reading files as dcecp scripts 2-46
reconfiguration, network 1-57
reconfiguring DTS 6-30
reconfiguring DTS temporarily 6-31
reconfiguring the registry 1-22
recovering slave replicas 7-110
recovering the registry 7-112
recovering, master replicas 7-111
registering cell names 1-8
registries
  changing the master replica site 7-37
  default ticket lifetime property 7-108
  deleting when machines are removed 7-125
  displaying and setting policies 7-35
  hidden password property 7-108
  local 7-92
  maintaining local 7-92
  maximum ticket lifetime 7-34
  maximum UNIX ID property 7-109
  minimum group ID property 7-108
  minimum organization ID property 7-109
  minimum ticket lifetime property 7-109
  minimum UNIX ID property 7-109
  password expiration date 7-105
  password format 7-105
  policies 7-104
registry
  extending 1-49
  using the dcecp program 1-21
registry ACL Manager 7-112
registry attribute 7-25
registry capacity property 7-92
registry connect command 7-61
registry connect command options 7-60
Registry Database
  accessing objects 1-44
  ACL Manager 7-112
  authentication information 7-116
  backing up 7-36
  creating 7-56
  initial ACLs 7-89
  management information 7-115
  physical security 7-103
  planning sites for components 7-103
  populating 7-106
  replication 7-125
  restoring 7-37
  setting up 7-139
  starting explicitly 7-141
Registry Database *(continued)*
  storage 7-88
  structure 7-142
  user information 7-117
  verifying 7-146
  viewing information 1-57
Registry Editor usage 7-113
registry entries
  group 7-134
  password 7-134
registry entries created by passwd_import 7-101
registry maintenance
  adding accounts 7-13
  backing up and restoring the registry database 7-36
  changing the registry master key 7-38
  consistent files 7-74
  overrides 7-95
registry objects, accessing 1-44
registry reconfiguration 1-22
Registry Service 1-44
reissuing errors 2-28
relative time 1-41
relocating a clearinghouse 5-57
remote administration machine programs 1-17
Remote Procedure Call (RPC) introduction 1-2
remote time sources 6-14
removing a host from a cell 3-14
removing binding information 4-24
removing expired credentials files 1-22
removing members from a group entry 4-27
removing members from a profile entry 4-30
renaming attribute types 7-51
replica, turning slave into master 7-111
replicas
  about 1-31
  changing master site 7-37
  creating 5-17
  creating slave (Security) 7-56
  deleting 5-25
  deleting master 5-24
  deleting secondary replicas (Security) 7-125
  excluding from replica set 5-51
  forcibly deleting secondaries (Security) 7-83
  master, designating 5-51
  modifying replica set 5-51
  recovering master 7-111
  recovering slave 7-110
  starting master (Security) 7-141
  turning master into slave 7-145
  verifying status 7-146
replicated databases 7-126
replication
  backing up namespaces 5-5
  cell namespace 1-11
  introduction 1-4

replist object definition 7-142
requirements for client machines 1-12
reserved accounts 7-129
reserved principals 7-129
resolving conflicts 7-102
resource records 5-31
restarting
  clerk processes 5-59
  primary server in locksmith mode 7-131
  Security server in locksmith mode 7-130
  security servers 7-130
  server processes 5-59
restoring
  duplicate masters 7-131
  merged CDS directories 5-60
  registry database 7-36
restructuring namespaces 1-33
retrieving server configuration information 4-5
rgy_edit commands properties 7-92
routing serviceability messages 4-33
RPC
  client programs 1-13
  defined 1-2
  interaction with directory service 5-25
  server programs 1-14
rpccp 1-13
runtime files A-4

S

sample passwd_import session 7-131
satellite, UTC A-9
schema 1-49
schema definitions for password synchronization 3-15
schema entries, displaying 7-73
script usage 2-40
scripts
  case command 2-2
  conditional statements 2-2
  controlling 2-1
  if statement 2-2
  informal administration 2-39
  loops 2-2
  running operating system commands 2-47
  testing for patterns 2-2
scripts, documenting 2-23
scripts, writing 1-25
sec-admin group 1-12
sec_admin commands
  administration programs 1-17
  changing the registry master key 7-38
  description 7-136
  for reconfiguring replica sets 7-137
sec_create_db 1-17
sec_create_db command 7-56
sec_create_db results 7-58
sec/group directory A-32
sec/group/subsys directory A-36
sec/principal directory A-40
sec/principal/hosts directory A-46
secd 7-103
secd locksmith option 7-131
security
  physical 7-103
  planning 1-11
security directory, top level A-29
security integration, AIX/DCE 7-24
security object names 7-135
Security Servers
  administration programs 1-17
  client programs 1-13
  control program for managing 1-47
  dcecp commands for managing 2-17
  maintenance tasks 1-21
  managing 1-21
  objects in dcecp 2-16
  registry backup procedures 3-1
  restarting 7-130
  restarting in locksmith mode 7-130
  starting in locksmith mode 7-140
Security Service commands
  kdestroy 7-88
  klist 7-86
security space A-27
server considerations 1-12
server entries in CDS
  creating 4-24
  deleting 4-24
server entry information, viewing 4-24
server entry names, unique 4-22
server epochs, matching 6-20
server machine files 1-14
server management operations access 5-14
server processes
  CDS 1-15
  Security 1-14
server profiles 4-27
server programs
  DFS 1-16
  DTS 1-16
serverentry attribute 6-26
servers
  accounts 7-137
  configuration needs 4-3
  creating 6-30
  defining attribute trigger 7-63
  deleting from network 7-125
  disabling 4-5
  DTS 6-9
  enabling 4-5
  extending configurations 4-6

servers *(continued)*
listing configuration information 4-5
modifying attributes 6-21
modifying configurations 4-6
monitoring counters 5-54
operation, controlling 4-3
querying 4-7
removing machines from network (Security) 1-57
restarting 5-59
restarting primary in locksmith mode 7-131
retrieving configuration information 4-5
starting 4-5
stopping 4-5
unconfiguring 4-5
servers required characteristic 6-10
Servers, DTS
configuring as NTP server 6-15
couriers 6-10
global 6-10
local 6-9
management 6-19
service servers, authentication 1-56
service tickets 7-86
set directory command 5-50
set directory to new epoch command 5-51
set directory to skulk command 5-64
set link command 5-44
setting
_s(sec) variable 7-138
access control 5-62
capacity on local registry 7-93
clerk attribute values 6-30
DCE Audit environment variables 7-138
lifespan on local registry 7-93
policies 7-106
properties 7-106
server attribute values 6-30
setting ticket lifetimes 7-11
show cell command 5-31
show clearinghouse command 5-28
simple name 5-3
simultaneous address changes 7-85
skew 1-36
skulk operation 5-63
skulks 1-19
slave replicas, recovering 7-110
soft links
changing destination name 5-44
changing expiration value 5-44
changing extension value 5-44
creating 5-43
deleting 5-44
description 1-32
example 5-67
managing 5-43
source command 2-46
spawning subprocesses 2-47
stability, cell 1-9
starting
Audit daemon 7-140
clearinghouses 5-58
configured DCE processes 3-20
DCE host services 4-13
dcecp 2-16
Registry Editor 7-113
Security server in locksmith mode 7-140
servers 4-5
starting master replicas 7-141
stopping
DCE host services 4-13
DCE processes 3-20
dcecp 2-16
existing clerk or server 6-30
Registry Editor 7-113
servers 4-5
storing registry databases 7-88
strings
constructing 2-42
manipulation 2-42
parsing 2-42
subprocesses, spawning 2-47
substituting commands 2-1
substituting variables 2-48
subtrees, merging 1-34
support, applications 6-5
synchronize command, dtscp 6-2
synchronizing CDS server clerks 5-64
synchronizing system clocks 1-37
syncinterval attribute 6-24
syntax, ACL 7-1
SYSTEM attribute 7-25
system time 6-1
system time, changing 6-1
system time, modifying 1-20

T

task objects
formal 2-29
model 2-30
parsing arguments 2-35
TDF
See Time Differential Factor
telephone services, UTC A-9
terminate loops 2-2
testing cell operation 3-21
testing for host activity 3-22
ticket cache name 7-87
ticket lifetime property 7-108
ticket-granting tickets 7-86

tickets
deleted passwords 3-6
setting lifetimes 7-11
Time Differential Factor 1-41
time providers
and drift 6-5
definition 1-35
dts_ntp_provider.c program 6-13
dts_null_provider.c program 6-13
planning for 6-11
selection criteria 1-60
support 6-5
time representation, DTS 1-40
time source, criteria for selecting A-2
time sources, local NTP 6-13
time sources, remote NTP 6-14
time zone map A-50
timestamps
Creation Time Stamp (CTS) 5-37
DTS 6-12
Update Timestamp (UTS) 5-37
timestamps, keeping data consistent 5-37
tolerance attribute 6-24
tools, looking beyond 2-41
top-level CDS directory A-13
trail file 1-54
transitive trust relationships 7-144
translating names to resources 5-65
traps 2-27
trigbind option 7-63
troubleshooting security 7-29
trust relationships
creating 7-60
direct 7-144
establishing 7-75
transitive 7-143

U

unauthenticated mask
effect on ACL checking 7-7
inappropriate use of 7-8
unconfiguring servers 4-5
unique server entry names 4-22
Universal Unique Identifier (UUID) 7-145
UNIX IDs
for foreign users 7-76
setting in registry 7-109
Universal Unique Identifier (UUID) 7-145
unreachable names 5-34
update propagation, CDS 5-69
Update Timestamp (UTS) 5-37
updating data in CDS 1-34
updating the pe_site file 7-84
user accounts 7-146
user information, displaying 3-11
user interface, CDS 5-11
user object, modifying 3-13
users, managing 1-27
using attribute sets 7-48
UTC (Coordinated Universal Time)
Network Time Protocol (NTP) A-9
radio transmissions A-9
satellite A-9
telephone services A-9
time intervals 1-35
UTC sources A-9
utilities, DCE administration 1-16
UTS
See Update Timestamp (UTS)
UUID
See Universal Unique Identifier (UUID)

V

valid characters A-48
validating authenticity of DCE Security Service 7-146
variable substitution 2-48
variables, environment 4-37
verifying replicas are running 7-146
viewing group entry members 4-27
viewing members of a profile entry 4-29
viewing server entry information 4-24

W

WAN cells 6-8
working with files 2-49
world map, time zones A-50
writing
dcecp objects 1-25
scripts 1-25

X

xattrschema object
access control 7-143
defined 7-142
displaying registry information 7-73
xntpd daemon 6-15