OS/390®

DCE
Administration Guide

SC28-1584-05

OS/390®

DCE Administration Guide

SC28-1584-05

Note

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

| **Sixth Edition (March 1999)**

| This edition, SC28-1584-05, applies to Version 2 Release 7 of OS/390 DCE Base Services, OS/390 DCE User Data Privacy (DES and CDMF), OS/390 DCE User Data Privacy (CDMF) (5647-A01), and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Changes are made periodically to the information herein.

For a list of changes, see “Summary of Changes” on page xxxiii.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

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

IBM welcomes your comments. A form for reader's comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
Information Development, Dept. G60
1701 North Street
Endicott, NY, 13760-5553
United States of America

FAX (United States & Canada): 1+607+752-2327
FAX (Other Countries):
Your International Access Code +1+607+752-2327

IBMLink™ (United States customers only): GDLVME(PUBRCF)
IBM Mail Exchange: USIB2L8Z@IBMMAIL
Internet e-mail: pubrcf@vnet.ibm.com
| World Wide Web: http://www.ibm.com/s390/os390/

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

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

The following statements are provided by the Open Software Foundation.

Contents

Notices . . . xxvii
Trademarks . . . xxvii
Websites in This Book . . . xxviii

About This Book . . . xxix
Who Should Use This Book . . . xxix
How to Use This Book . . . xxix
Conventions Used in This Book . . . xxix
Product Name . . . xxx
Where to Find More Information . . . xxx
Online Books . . . xxxi

Summary of Changes . . . xxxiii

Part 1. DCE Administration Concepts . . . 1

Chapter 1. Introduction to DCE System Administration . . . 3
Clients and Servers . . . 3
Remote Procedure Call . . . 4
Cells . . . 4
The DCE Namespace . . . 5
Filespaces . . . 5
Principals . . . 5
Access Control Lists . . . 6
Caching . . . 6
Replication . . . 6

Chapter 2. Overview of DCE Components . . . 7
Description of DCE Components . . . 8
  DCE Remote Procedure Call (RPC) . . . 8
  DCE Directory Service . . . 9
  DCE Distributed Time Service (DTS) . . . 12
  DCE Security Service . . . 14
How the DCE Components Work Together . . . 15
  Remote Procedure Call . . . 16
  Directory Service . . . 16
  Distributed Time Service . . . 16
  Security Service . . . 17
What Makes Up DCE System Administration? . . . 17
  Summary of DCE Daemons and Administrative Facilities . . . 17

Chapter 3. Overview of DCE Maintenance . . . 19
RPC Maintenance Tasks . . . 19
  Maintaining NSI Entries . . . 19
  Maintaining the Endpoint Map . . . 19
  Controlling Access to the Endpoint Map . . . 19
| Restricting Network Interfaces Used by DCE . . . 20
Cell Directory Service Maintenance Tasks . . . 20
  Monitoring the Cell Directory Service . . . 20
  Managing the Cell Directory Service . . . 21

Cell Directory Service Security and Access Control . . . 21
Distributed Time Service Maintenance Tasks . . . 22
Managing the Distributed Time Service . . . 22
Modifying the System Time . . . 23
DCE Security Service Maintenance Tasks . . . 23
Reconfiguring the Registry . . . 24
Improving Registry Performance in Large Cells . . . 24
| Improving dce_login Performance . . . 25

Part 2. General Administration Aspects of OS/390 DCE . . . 27

Chapter 4. OS/390 DCE Administration Commands . . . 29
Using the OS/2® Administration GUI . . . 29
DCE Daemons in OS/390 DCE . . . 29
Administration and User Commands . . . 29
Entering Arguments to OS/390 DCE Commands . . . 30
Running the Administration and User Commands in Batch . . . 31
Commands that Cannot Fit in One Line . . . 32
Command Input and Output Redirection . . . 32

Chapter 5. Starting and Stopping OS/390 DCE . . . 33
DCEKERN Address Space . . . 33
Stopping DCEKERN . . . 34
Who Can Start and Stop OS/390 DCE Daemons? . . . 34
Ways of Starting OS/390 DCE Daemons . . . 34
Using DCECONF to Start OS/390 DCE . . . 35
The MODIFY DCEKERN Operator Command . . . 35
Using MODIFY DCEKERN to Start OS/390 DCE Daemons . . . 35
Order of Starting OS/390 DCE Daemons . . . 35
Using MODIFY DCEKERN to Stop OS/390 DCE Daemons . . . 36
Viewing the Status of DCE Daemons . . . 36
Starting OS/390 DCE During System IPL . . . 37
Daemon Configuration File . . . 37
How DCEKERN Starts the DCE Daemons . . . 38
Using the -nodce Option to Start DCEKERN . . . 38
Stopping and Restarting the CDS Advertiser and Clerk Daemons . . . 38
Controlling Where Daemons are Started . . . 39
Using the Control Task to Monitor HFS Utilization by DCE . . . 39

Chapter 6. Structure and Backup of OS/390 DCE Files . . . 41
Global Files . . . 41
Local Files . . . 41
Backup Strategy . . . 42
Utilities for Backing Up Files . . . 42
Mapping of HFS to PDS . . . 42

Part 3. The DCE Control Program . . . 43

Chapter 7. DCE Control Program Introduction . . . 45
Flexible, Portable, and Extensible Administration . . . 45
DCE Administration Objects . . . 46
Using the DCE Control Program . . . 47

Starting and Stopping the dcecp Program 47
Invoking dcecp Operations 47
Disabling the Alternate operation object Syntax 49
Using Global Error Information Variables 49
Doing More with dcecp 50
When To Use an Interactive Command or Script 52
Editing Command Lines 52
Editing Command Lines with the history Command 53
Using the dcecp Help Facilities 54
Customizing dcecp Sessions 56
Adding Scripts to dcecp Sessions 57
Adding New Objects to the DCE Control Program 59
Convenience Variables Mean Fewer Keystrokes 59
Current Principal (User) Name (_u) 60
Current Cell Name (_c) 60
Current Host Name (_h) 60
Most Recent Operation Argument Name (_n) 60
Parent of the _n (_p) 61
Last dcecp Object Name (_o) 61
Last Operation's Return Value (_r) 62
DCE Servers to Use (_s(xxx)) 63
Last Security Server Used (_b(sec)) 64
The proc Command Lets You Create New Commands 64
Running Operating System Commands from a Script 66

Chapter 8. Writing Scripts and dcecp Objects 69
Informal Administration Scripts 69
Formal Task Objects 71
A Model for Task Objects 71
Using the parseargs Procedure 78
Invoking Task Objects 80

Part 4. DCE Administration Tasks 81

Chapter 9. DCE Administration Task Objects 83
Using Task Objects to Simplify DCE Administration 83
Getting Started with Objects 84
Looking Beyond the Tools 84

Chapter 10. Managing a DCE Cell 85
Showing All Configured DCE Servers and DCE Hosts 85
Testing Cell Operation 86
Modifying or Extending the Cell Object 87
| Using the eNetwork Communications Server with OS/390 DCE 87

Chapter 11. Managing DCE Hosts 89
Listing the DCE Hosts in a Cell 89
Showing All Servers Configured for a DCE Host 90
Testing Whether a DCE Host Is Running 90
Starting Configured DCE Processes on a Host 91
Stopping DCE Processes Running on a Host 91
Modifying or Extending the Host Object 91

Chapter 12. Managing DCE Users 93
Creating a New User 93
Showing User Information 94
Deleting a User 95
Modifying or Extending the User Object 96

Part 5. DCE Host and Application Administration 97

Chapter 13. Managing DCE Host Services and Host Data 99
DCE Host Services 99
  Starting and Stopping DCE Host Services 100
Managing Host Data 100
  Permissions For Accessing Host Data 101
  Modifying Host Cell Name Information 102
  Manipulating Data in Other Host Files 103

Chapter 14. DCE Application Administration 105
Controlling Server Operation 105
  Common Server Configuration Needs 106
  Configuring Servers 111
  Listing and Retrieving Server Configuration Information 112
  Unconfiguring Servers 112
  Starting and Stopping Servers 112
  Disabling and Enabling Services 113
  Extending Server Configurations 113
  Changing Server Configurations 115
  Checking Whether Servers Are Running 116
Managing Client-Server Binding Information 116
The Endpoint Map Eases Application Development and Administration 118
  The Endpoint Map 118
  Endpoint Map Administration is Mostly Automatic 118
  Commands for Monitoring the Endpoint Map 119
  Commands for Rebuilding the Endpoint Map 119
  Command to Determine if a Server is Listening 120
Other Endpoint Map Administration Tasks 120
  Deleting the Endpoint Map During Startup 120
  Recovering the Endpoint Map 120
  Restricting Endpoints 121
  Viewing Information in the Endpoint Map 122
Managing Server Entries, Groups, and Profiles in CDS 123
  The RPC_DEFAULT_ENTRY Environment Variable 123
  Unique Server Entry Names Identify Individual Servers and Objects 123
  Group Entries Help Balance Server Workloads 127
  Profiles Help Direct Clients' Searches For Servers 130
Client Administration 135
  Determining the Entry Name 136
  Providing the Entry Name to Clients 136

Chapter 15. Examples of Setting Up RPC Profiles 139
Math Server Example 139
  Frequency of Use 139
  Solution 140
  Steps in Creating the RPC Profile 140

Print Server Example 142
  Frequency of Use 143
  Solution 143
  Steps in Creating the RPC Profile 143

Chapter 16. Controlling Access to the DCED Endpoint Map 149
Overview of DCED Endpoint Map Authorization 149
  ACL Entry Types Supported by the DCE Endpoint Map ACL Manager 149
  Permissions Used for the DCE Daemon Endpoint Map 149
DCE Clients' Access to DCED 150
Giving Application Servers Access to the DCED Endpoint Map 150
  RPC Server Group 150
Giving DCE Administrators Access to the DCED Endpoint Map 151
The Default Endpoint Map ACL 152
Giving Unauthenticated Users Access to the DCED Endpoint Map 152

Chapter 17. Workload Balancing in a Parallel Sysplex Environment 153
Overview of Workload Balancing 153
Setting Up Workload Balancing 154
  Server Requirements 154
  Client Requirements 154
Using Workload Balancing 154
  Using the Environment Variable 154
  Using the Application Programming Interfaces 155
  Using dcecp endpoint Commands with Workload Balancing 155
  The PI Program 155

Part 6. DCE Directory Service 157

Chapter 18. Introduction to the DCE Directory Service 161
How the DCE Components Use Directory Services 161
How to Use Directory Service 162
Directory Services and the Cell Environment 163
How Cells Determine Naming Environments 165
  Global Names 165
  Cell-Relative Naming in a Standalone Cell 166
  Local Filenames 167
A Closer Look at DCE Names 167
  Cell Directory Service Names 167
  Global Directory Service Names 168
  Domain Name System Names 170
  Names Outside of the DCE Directory Service 171

Chapter 19. Cell Directory Service Concepts 173
How the Cell Directory Service Works 173
Replicas and Their Contents 175
  Object Entries 176
  Soft Links 176
  Child Pointers 177
  Putting It All Together 177
CDS Advertiser 177
CDS Advertiser and Clerk in OSF DCE 178
CDS Advertiser and Clerk in OS/390 DCE 179

Security in the Cell Directory Environment 179
Protecting the CDS Cache File 181
Conversion between ASCII and EBCDIC in OS/390 DCE 181
Cell Directory Service User Interface 181

Chapter 20. How the Cell Directory Service Looks Up Names 183
Translating from Names to Resources 183
How the Cell Directory Service Finds Names 187
  The Solicitation and Advertisement Protocol 188
  Lookups 188
  The cdscache create Command 188

Chapter 21. How the Cell Directory Service Updates Data 191
Update Propagation 191
Skulk Operation 191
How Timestamps Help Keep Data Consistent 192

Chapter 22. Managing the DCE Directory Service 193
Using the dcecp Program 193
  CDS Managed Objects 193
  dcecp Command Operations for CDS 193
  CDS Object Attributes 194
Using the cdscp Program 194
  Starting and Exiting 194
  The cdscp Program Commands 195

Chapter 23. Controlling Access to CDS Names 197
Overview of DCE Authorization for the Cell Directory Service 197
ACL Types Supported by the Cell Directory Service 197
How Permissions Propagate to CDS Directories and Their Contents 198
Access Control List Entry Types 198
DCE Permissions Supported by the Cell Directory Service 200
Controlling Access to CDS Clerk and Server Management Operations 200
Control Program Commands and Required Permissions 201
Editing ACLs on Cell Directory Service Names 203
How CDS Servers Gain Access to the Namespace 204
Setting Up Access Control in a New Namespace 204
  Adding Members to the Namespace Authorization Group 204
  Creating Additional Authorization Groups 205
  Establishing Maximum Permissions for Unauthenticated Principals 205

Chapter 24. Managing Clerks, Servers, and Clearinghouses 207
Monitoring Clerk, Server, and Clearinghouse Counters 207
  Displaying Clerk Counters 207
  Displaying Server Counters 207
  Displaying Clearinghouse Counters 207
Monitoring Clerk Communication with Specific Clearinghouses 207
Displaying the Contents of a Clearinghouse 208
Forcing the Clearinghouse to Checkpoint to Disk 208
Disabling Clerks and Servers 208
  Disabling a Clerk 208
  Disabling a Server 208
Stopping the CDS Advertiser and CDS Clerk 209
Starting and Stopping the CDS Daemon 209

Restarting the CDS Advertiser and CDS Clerk . . . 209
Automatic Restart after Abnormal Termination . . . 210
Removing Stale Cache Entries . . . 210
Recovering from a Corrupted CDS Cache . . . 211
If Cache Size Is Changed . . . 211
Preserving a Clearinghouse across a Server System Upgrade . . . 211
Recovering from a cdsd File System Full Condition . . . 212
Reconfiguring a Secondary CDS Server After Deconfiguring . . . 216
Backing Up Namespace Information . . . 217
Using Replication to Back Up Namespace Information . . . 217
Using Operating System Backups . . . 217

Chapter 25. Managing CDS Directories . . . 219
Creating a Directory . . . 219
Permissions for Creating a Directory . . . 219
Entering the directory create Command . . . 219
Checking the ACL Entries for a New Directory . . . 220
Upgrading the Directory Version on the Cell Root Directory . . . 220
Upgrading the Directory Version on a Directory . . . 220
Creating a Read-Only Replica . . . 221
Before You Create a Replica . . . 221
Permissions for Creating Replicas . . . 223
Entering the directory create Command . . . 223
Deleting a Read-Only Replica . . . 223
Permissions for Deleting a Replica . . . 223
Entering the directory delete Command . . . 223
Skulking a Directory . . . 224
Permissions for Skulking a Directory . . . 224
Entering the directory synchronize Command . . . 224
Synchronizing CDS Server Clocks . . . 225
Changing a Directory's Convergence . . . 225
Directory Convergence Set to High . . . 225
Directory Convergence Set to Medium . . . 225
Directory Convergence Set to Low . . . 226
Before You Modify a Directory's Convergence . . . 226
Permissions for Modifying a Directory's Convergence . . . 226
Entering the directory modify Command . . . 226

Chapter 26. Viewing the Structure and Contents of a Namespace . . . 227
Listing the Contents of Directories . . . 227
Displaying the Attribute Values of CDS Names . . . 227
Displaying Clerk and Server Attribute Information . . . 229
Preferred Clearinghouse for Viewing the Namespace . . . 229

Chapter 27. Using the CDS Subtree Commands to Restructure CDS Directories . . . 231
Overview of the Merge and Append Procedures . . . 231
Merging CDS Directories . . . 232
Appending CDS Directories . . . 233
Modifying ACLs at the Target Location . . . 234
Handling Errors . . . 235
Duplicate Names . . . 235
Unreachable Name Failures . . . 235
Insufficient Permissions . . . 235
Merging CDS Directories into a Foreign Cell . . . 236

Establishing Cross-Cell Authentication 236
Performing a Merge Operation into a Foreign Cell 236
Restoring Merged CDS Directories 236

Chapter 28. Restructuring a Namespace 239
Managing Soft Links 239
Creating a Soft Link 239
Setting Expiration and Extension Values for a Soft Link 239
Changing a Soft Link Destination Name 240
Changing a Soft Link Expiration or Extension Value 240
Deleting a Soft Link 241
Changing a Directory Replica Set 241
Before You Change a Directory's Replica Set 242
Permissions Required for Modifying a Replica Set 242
Designating a New Master Replica 242
Excluding a Replica from a Replica Set 243
Deleting Directories 244
Deleting a Non-Replicated Directory 245
Deleting a Directory Replica 245
Relocating a Clearinghouse 246
Dissociating a Clearinghouse from Its Host Server System 247
Copying the Clearinghouse Database Files to the Target Server System 247
Starting the Clearinghouse on the Target Server 248
Deleting a Clearinghouse 249
Before You Delete a Clearinghouse 249
Permissions for Deleting a Clearinghouse 249
To Delete a Clearinghouse 250

Chapter 29. Managing Intercell Naming 251
How the Global Directory Agent Works 251
Managing the Global Directory Agent 254
Enabling Other Cells to Find Your Cell 254
Defining a Cell in the Domain Name Service 255
Defining a Cell in an LDAP Server 256

Part 7. DCE Distributed Time Service 257

Chapter 30. Introduction to the DCE Distributed Time Service 259
Distributed Time Service Advantages 260
Applications Support 260
External Time Provider Support 260
Manageability 261
Quantitative Inaccuracy Measurement 261
Basic Distributed Time Service Concepts 262
Time Measurement Factors 262
Inaccuracy Values 262
Synchronizing System Clocks 263
How the Distributed Time Service Adjusts System Clocks 265
Distributed Time Service Time Representation 265
How DTS Works 268
Clerks 268
Servers 269

Chapter 31. Managing the Distributed Time Service 271
Using the dcecp Program 271
DTS Objects 271
DTS Command Operations 271
DTS Object Attributes and Counters 272
DTS Timestamp Format 273
Reconfiguring DTS on Nodes 274
Stopping an Existing Clerk or Server 275
Creating a New Clerk or Server 275
Setting Clerk and Server Attribute Values 275
Modifying Clerk and Server Attributes 276
The minservers Attribute 277
Use of minservers Attribute with Global Servers 278
Use of minservers Attribute with Systems on Point-to-Point Lines 279
The maxinaccuracy Attribute 279
The syncinterval Attribute 280
The tolerance Attribute 281
The localtimeout, globaltimeout, and queryattempts Attributes 282
The serverentry and serverprincipal Attributes 282
Management Tasks Specific to Servers 283
Designating Global and Courier Servers 283
Designating Global Servers Outside a Cell 284
Matching Server Epochs 285
Setting the checkinterval Attribute for Connection to a Time Provider 285
Changing the System Time 286
Updating the Time Monotonically 286
Updating the Time Nonmonotonically 286
Forcing System Synchronization 287
Controlling Access to DTS 287

Chapter 32. OS/390 DCE Considerations in DTS 289
DCE Software Clock 289
Null Time Provider 289
What Is the Null Time Provider Program? 289
DTS and the OS/390 Sysplex Environment 290
Setting the Time Zone 290
Overriding the System Time Zone 291
Using POSIX Time Format in TZ 291
Resetting the DCE Software Clock 291

Part 8. DCE Security Service 293

Chapter 33. Overview of the DCE Security Service 297
DCE Authentication Servers and Clients 297
The Registry Database 298
Cells 299
The Logical Identities of Principals, Groups, and Organizations 299
Full Pathnames of DCE Objects 299
Full Pathnames for Registry Objects 300
Physical Security of the Database 300
How the Registry Database is Stored 300
Replicated Databases 301
How Updates Are Handled 301

Master and Slave Replicas . . . 302
Handling Database Updates . . . 303
Propagating Database Changes . . . 304
Master/Slave Authentication . . . 304
Names for Security Objects . . . 304
Using Names with dcecp Security Service Commands . . . 305
Using Names with the dcecp acl Command . . . 305

Chapter 34. Using Access Control Lists . . . 307
Authorization Overview . . . 307
ACL Managers . . . 308
ACL Interpretation . . . 309
Privilege Attributes Inherited by Processes . . . 309
ACL Entries and Masks . . . 309
ACL Syntax . . . 310
ACL Entry Types for Principals and Groups . . . 310
Group Permissions and Project Lists . . . 314
Using Principal and Group ACL Entries . . . 314
ACL Entry Types for Masks . . . 315
ACL Entry Types for Dissimilar DCE Releases . . . 315
The Checking Sequence for ACL Entries . . . 316
Denying Access . . . 319
ACL Management Tasks . . . 319
Copying ACLs . . . 320
Generating ACLs from Files . . . 320
Container ACLs . . . 321
Objects and Containers . . . 321
Initial ACLs for Objects and Containers . . . 321
Effect of Masks when Editing ACLs . . . 324

Chapter 35. Control Programs for Managing the Security Service . . . 327
Using the DCE Control Program . . . 327
Security Service Objects . . . 327
Security Service Command Operations . . . 328
Using the Registry Editor . . . 329
Starting, Stopping, and Getting Help . . . 329
Using the sec_admin Program . . . 330
Starting, Stopping, and Getting Help . . . 330
The sec_admin Commands for Reconfiguring Replica Sets . . . 330

Chapter 36. Creating and Maintaining Principals, Groups, and Organizations . . . 331
Principal, Group, and Organization Names . . . 331
Primary Names . . . 331
Full Names . . . 331
Aliases . . . 332
Name Formats . . . 332
Reserved Principals and Accounts . . . 332
Object Creation Quotas . . . 333
Universal Unique Identifiers and UNIX IDs . . . 333
Adding and Maintaining Principals . . . 334
Adding Principals . . . 334
Changing Principals . . . 335
Deleting Principals and Aliases . . . 336
Extended Security Attributes for Principals . . . 336

OS/390 DCE Release 1 Authentication . . . 336
Managing Logins that are Not Valid . . . 339
Managing Password Strength and Password Generation . . . 339
Adding and Maintaining Groups and Organizations . . . 341
Project Lists . . . 341
Adding Groups and Organizations . . . 342
Changing Groups and Organizations . . . 343
Deleting Groups and Organizations . . . 343
Maintaining Membership Lists . . . 344
Effects of Account Creation on Membership Lists . . . 344
Example: Adding and Deleting Group Members . . . 344
Creating and Maintaining Aliases for Principals or Groups . . . 345
Creating Aliases . . . 345
Changing Primary Names to Aliases and Vice Versa . . . 345

Chapter 37. Creating and Maintaining Accounts . . . 347
User Accounts . . . 347
Server Accounts . . . 347
Passwords for Server Accounts . . . 348
Accounts for Servers . . . 348
Steps for Creating Server Accounts . . . 348
Machine Accounts . . . 349
How Identities Represented by Accounts Are Authenticated . . . 349
Privilege Attributes . . . 349
Ticket-Granting Tickets and Tickets to Services . . . 350
Displaying Privilege Attributes and Tickets . . . 350
Destroying a Principal's Tickets . . . 351
Adding Accounts . . . 351
Setting Ticket Lifetimes . . . 353
Adding Accounts Example . . . 354
Modifying Accounts . . . 355
Deleting Accounts . . . 355
Reactivating Accounts that have Expired . . . 356
Creating, Maintaining, and Deleting Keytab Files . . . 357
The Keytab File . . . 357
Creating and Maintaining Keys and Keytab Files . . . 358
Removing Keytab Files . . . 360
Changing Server and Machine Passwords in the Keytab File . . . 361
Handling Compromised Server or Machine Passwords in the Keytab File . . . 361

Chapter 38. Creating and Using Extended Registry Attributes . . . 363
The xattrschema Object . . . 363
Creating and Maintaining Attribute Types . . . 363
Creating Attribute Types . . . 363
Modifying Attribute Types . . . 365
Renaming Attribute Types . . . 365
Deleting Attribute Types . . . 365
Defining the ACL Managers for Attributes . . . 366
Defining the Attribute Type Encoding . . . 367
Defining Attribute Trigger Servers . . . 368
The trigtype Option . . . 368
The -trigbind Option . . . 369
Creating and Maintaining Attribute Instances . . . 371
Attaching Attribute Instances to Objects . . . 371

Modifying Attribute Instances 371
Deleting Attribute Instances 372
Using Attribute Sets 373

Chapter 39. Administering a Multicell Environment 375
Trust Relationships 375
Direct Trust Relationships 375
Establishing Trust Relationships 376
Creating Trust Relationships 376
Command Options for the registry connect Command 377
The Accounts Created by the registry connect Command 378
Changing Cross-Cell Authentication Accounts 379

Chapter 40. Viewing Registry Information 381
Displaying Account Information 381
Displaying Group and Organization Information 382
Displaying Principal Information 384
Displaying xattrschema Information 385
Displaying ACL Information 386
Displaying keytab Information 387

Chapter 41. RACF Interoperability and Single Sign-on 389
Overview of RACF Interoperability 389
The RACF Interoperability Utilities 390
Tailoring the Utilities for Your Environment 393
Tailoring Variables Common to mvsimpt and mvsexpt 393
Tailoring Variables for mvsimpt 393
Tailoring Variables for mvsexpt 394
Guidelines for Using mvsimpt and mvsexpt 395
Multi-Cell Considerations 398
Cross Linking Small Numbers of Users 400
Recreating the Processed Entries File 400
Introduction to Administration Scenarios 400
Cross Linking Existing RACF Users who are New DCE Users 400
Cross Linking Existing DCE Users who are New RACF Users 405
Cross Linking Existing DCE Users who are Existing RACF Users 407
Single Sign-on for OS/390 and DCE 409
Preparing for DCE Single Sign-on 409
Automatic DCE Single Sign-on Invocation 410
User Control of Automatic DCE Single Sign-on 410

Chapter 42. Maintaining Policies and Properties 411
Policies 411
Standard Policy 411
Authentication Policy 413
Handling Conflicting Policies 414
The Effects of Changes on Existing Policies 414
Displaying and Setting Standard and Authentication Policies 415
Properties 415
Default Ticket Lifetime Property 415
Hidden Password Property 415
Minimum Group ID Property 416
Minimum Organization ID Property 416
Minimum UNIX ID Property 416

Maximum UNIX ID Property . . . 416
Minimum Ticket Lifetime Property . . . 416
Displaying and Setting Properties . . . 417

Chapter 43. Performing Routine Maintenance . . . 419
Adding Accounts . . . 419
Changing the Registry's Master Key . . . 419
Validating the Authenticity of the DCE Security Service . . . 420
Backing Up and Restoring the Registry Database . . . 420
Procedures for Backing Up the Registry Database . . . 420
Procedure for Restoring the Registry Database . . . 421
Setting the _s(sec) Variable . . . 422

Chapter 44. Handling Network Reconfigurations . . . 423
Changing the Master Replica Site . . . 423
Removing a Server Machine from the Network . . . 424
Handling Network Address Changes . . . 424
Updating the pe_site File . . . 424
Handling Simultaneous Address Changes . . . 425

Chapter 45. Adopting Registry Orphans . . . 427
What are Orphans? . . . 427
Solving the Problem of Orphans . . . 427

Chapter 46. Accessing Registry Objects . . . 429
The Registry Database . . . 429
Registry Permissions . . . 430
Management, Authentication, and User Information . . . 430
Permissions to Create Principals, Groups, or Organizations . . . 432
Permissions to Delete Principals, Group, or Organizations . . . 433
Permissions to Add Accounts . . . 433
Permissions to Delete Accounts . . . 436
Permissions to Add Members to Groups . . . 437
Permissions to Add Members to Organizations . . . 437
Permissions to Delete Members from Groups or Organizations . . . 437
Permissions to Change a Principal's, Group's, or Organization's Full Name . . . 438
Permissions to Change Management Information for Principals, Groups, or Organizations . . . 438
Permissions to Change Management, Authentication, and User Information (Except Passwords) for Accounts . . . 439
Permissions to Change Passwords for Accounts . . . 439
Permissions to Change Authentication and Management Information for Registry Policies and Properties . . . 439
Permissions to Run Commands that Act on Replicas . . . 440
Permissions to Create Extended Registry Attribute Types . . . 440
Permissions to Delete Extended Attribute Types . . . 440
Permissions to Modify Extended Registry Attribute Types . . . 441
Permission to Change ACLs on Registry Objects . . . 441
Permissions Required by Slave Replicas . . . 442
Registry ACL Manager . . . 442
Initial Registry ACLs . . . 443

Chapter 47. DCE Audit Service . . . 445
Features of the DCE Audit Service . . . 445
Components of DCE Audit Service . . . 445

DCE Audit Service Concepts 445
Audit Clients 446
Code Point 446
Audit Event 446
Event Number 446
Event Class 446
Event Class Files 447
Event Class Names 447
Event Class Numbers 447
Event Class Number Formats 448
Filters 448
Audit Trail File 451
Administration and Programming in DCE Audit 451
Programmer Tasks 452
Administrator Tasks 453

Chapter 48. DCE Audit Service Administrative Tasks 457
Setting DCE Audit Environment Variables 457
Starting the Audit Daemon 457
Controlling Access to the Audit Daemon 458
DCE Permissions Supported by the DCE Audit Service 458
Initial ACL of the Audit Daemon 458
Giving Permissions to Audit Clients and Administrators 458
Defining Event Classes 459
Steps in Defining an Event Class 459
Example Event Class File 459
Creating and Maintaining Filters 460
How To Create Filters 460
How To Modify Filters 461
How To Delete Filters 461
Default Filters 461
Enabling Audit Filters 462
Enabling and Disabling the Audit Logging Service 463
Modifying and Querying Audit Daemon Attributes 463
Controlling and Displaying Audit Trails 463
Displaying Audit Trail Files 463
Controlling the Audit Trail Size 464
Changing the Audit Trail File Storage Option 464

Chapter 49. Hardware Cryptography in DCE 465

Appendix A. Environment Variables in OS/390 DCE 467
Table of Environment Variables 467
Format for Setting Environment Variables 483
How to Set Environment Variables 483
Setting Variables in the Environment Variable File 483
Setting Environment Variables from the Shell 484
Setting Environment Variables from Batch or TSO 484
OS/390 DCE Daemon Environment Variable Files 485
Messaging Subsystem Environment Variables 485

Appendix B. The Code Set Registry 487
Character Sets and Code Sets 487
Code Sets and DCE 487

What is the DCE Code Set Registry? 487
The DCE Code Set Registry in OS/390 487

Appendix C. The DCE Cell Namespace 499
The Cell Directory Service Namespace 499
  The Top-Level CDS Directory 500
  The CDS hosts Directory 503
  The CDS subsys Directory 506
The Security Namespace 509
  The Top-Level Security Directory 509
  The sec/group Directory 512
  The sec/group/subsys Directory 516
  The sec/principal Directory 520
  The sec/principal/hosts Directory 525

Appendix D. Valid Characters and Naming Rules for CDS 529
Metacharacters 531
Additional Rules 531
Maximum Name Sizes 533

Appendix E. Object Identifier Files 535
Origin of Object Identifiers 535
CDS Attributes File 535
CDS Globalnames File 536
Modifying the Files 538

Appendix F. DTS Extended BNF 539
DTS Format Rules 539

Appendix G. Files Created and Used by mvsimpt and mvsexpt 541
Files Common to mvsimpt and mvsexpt 541
Files for mvsimpt 541
Files for mvsexpt 542
Examples of Files Created or Used by mvsimpt and mvsexpt 543
  The /opt/dcelocal/var/security/adm/DCEERS file 543
  The /opt/dcelocal/var/security/adm/PROCENTR file 543
  Layout of the /opt/dcelocal/var/security/adm/PROCENTR file 543
  Format of the /opt/dcelocal/var/security/adm/PROCENTR file 544
Examples of Files Created or Used by mvsimpt 545
  The /opt/dcelocal/etc/IMPTVAR file 545
  The /opt/dcelocal/var/security/adm/RACFUNLD file 546
  Layout of the /opt/dcelocal/var/security/adm/RACFUNLD file 546
  Format of the /opt/dcelocal/var/security/adm/RACFUNLD file 547
  The /opt/dcelocal/var/security/adm/DCEWORK file 547
  The /opt/dcelocal/var/security/adm/DCENEW file 547
Examples of Files Created or Used by mvsexpt 548
  The /opt/dcelocal/etc/EXPTVAR file 548
  The /opt/dcelocal/var/security/adm/RACFWORK file 550
  The /opt/dcelocal/var/security/adm/RACFERS file 550
  The /opt/dcelocal/var/security/adm/RACFNEW file 551
  The /opt/dcelocal/var/security/adm/ASUIDMAP file 551
  Format of /opt/dcelocal/var/security/adm/ASUIDMAP file 551
  The /opt/dcelocal/var/security/adm/PRNIDMAP file 552
  Format of the /opt/dcelocal/var/security/adm/PRNIDMAP file 552

Appendix H. DCE Security Administration Files 553
Auditable Events for the Audit Services 553
  Related Information 555
Auditable Events for the Time Services 555
  Related Information 559
Auditable Events for the Security Service 560
  Related Information 572

Glossary 573

Bibliography 591
OS/390 DCE Publications 591
OS/390 Security Server Publications 592
Tool Control Language Publication 592
IBM C/C++ Language Publication 592
OS/390 DCE Application Support Publications 592
Encina Publications 593

Index 595

Figures

1. Interaction of Clients and Servers . . . 4
2. OS/390 DCE . . . 7
3. CDS Requests a Name in Another Cell . . . 11
4. Location Information Is Returned to CDS . . . 11
5. CDS Contacts Another Cell . . . 12
6. DTS Relationships . . . 13
7. The DCEKERN Address Space . . . 33
8. Daemon Configuration File . . . 37
9. Server Binding Information . . . 117
10. Possible Information in a Server Entry . . . 124
11. Possible Mappings of a Group . . . 127
12. Possible Mappings of a Profile . . . 132
13. Example: Math Servers . . . 139
14. MathProf Search Path . . . 141
15. Print Servers . . . 142
16. Search Path for Print Servers . . . 147
17. Cell and Global Naming Environments . . . 163
18. Interaction of CDS, GDAs, and Global Directory Services . . . 164
19. Example of a Global Name . . . 165
20. Global Name of a Cell in DNS . . . 166
21. Sample CDS Namespace Hierarchy . . . 167
22. RDNs and Distinguished Names . . . 169
23. Comparison of CDS and GDS Names . . . 170
24. Sample Portion of the BIND Namespace . . . 171
25. Parts of a Global Principal Name . . . 171
26. Parts of a Global DCE File Name . . . 172
27. CDS Clerks and Servers on a LAN . . . 174
28. A Sample CDS Lookup . . . 175
29. Components of a CDS Server Node . . . 177
30. Interaction between the CDS Client, Clerk, and Advertiser in OSF DCE . . . 178
31. Interaction between the CDS Client, Clerk, and Advertiser in OS/390 DCE . . . 179
32. Logical and Physical Views of a Namespace . . . 184
33. Clearinghouse Object Entries and Clearinghouses . . . 185
34. A Soft Link and Its Resolution . . . 186
35. Child Pointers and Directories . . . 187
36. How the Clerk Finds a Name . . . 189
37. Example Namespace Hierarchy . . . 231
38. Example Namespace Before and After the Merge Operation . . . 233
39. Example Namespace Before and After the Append Operation . . . 234
40. Example Replica Set . . . 243
41. Example Replica Set After Master Redesignation . . . 243
42. Example Replica Set After Replica Exclusion . . . 244
43. How the CDS Clerk Finds a GDA . . . 252
44. How the GDA Helps CDS Find a Name . . . 253
45. Time and Inaccuracy . . . 263
46. Computed Time . . . 264
47. Adjustment of the Clock . . . 265
48. ISO-Compliant Time Format . . . 266
49. ISO-Compliant Time Format Variation . . . 267
50. Relative Time Format . . . 268

51. DTS Timestamp Format 274
52. Local Fault 281
53. Sysplex Timer-DTS Configuration 290
54. Machines, Servers, and the Database 298
55. Disk Memory and Virtual Memory Copies of the Registry Database 301
56. The Master Replica Update Process 302
57. Slave Replica Update Process 303
58. ACL Managers in Servers 308
59. Sample ACL Entries 310
60. Order of Checking ACLs and Applying Masks 318
61. Initial ACLs for Objects Created in Containers 322
62. Initial ACLs for Containers Created in Containers 323
63. Cross-Linking Utilities 391
64. Detailed View of RACF-DCE Cross-Linking Utilities 401
65. Registry Database Structure 429
66. Permission Required To Create Principals, Groups, or Organizations 432
67. Permissions Required To Delete Principals, Groups, or Organizations 433
68. Permissions Required to Add an Account and the Account Principal to the Group and Organization 434
69. Adding an Account for which the Principal is Already a Member of the Group and Organization 435
70. Permissions to Add an Account and the Principal to the Group Only 435
71. Permissions to Add an Account and the Principal to the Organization Only 436
72. Permissions Required to Delete Accounts 436
73. Permissions Required to Add Members to Groups 437
74. Permissions Required to Add Members to Organizations 437
75. Permissions to Delete Members from Groups or Organizations 438
76. Permissions Required to Change a Principal's, Group's, or Organization's Full Name 438
77. Permissions Required to Change Management Information for Principals, Groups, or Organizations 438
78. Permissions Required to Change Management, Authentication, and User Information (Except Passwords) for Accounts 439
79. Permissions Required to Change Passwords for Accounts 439
80. Permissions Required to Change Authentication and Management Information for Registry Policies and Properties 440
81. Permissions Required to Run Commands that Act on Replicas 440
82. Permissions Required To Create Extended Registry Attribute Types 440
83. Permissions Required To Delete Extended Registry Attribute Types 441
84. Permissions Required To View Extended Registry Attributes 441
85. Permissions Required To Modify Extended Registry Attribute Types 441
86. Permission Required to Change ACLs on Registry Objects 441
87. Event Class Number formats 448
88. Override Relations between Filter Types 451
89. Example Environment Variable File 483
90. The Top of the CDS Namespace and the SUBSYS Directory 499
91. The HOSTS Directory in The CDS Namespace 500
92. Valid Characters in CDS, GDS, and DNS Names 530
93. The /opt/dcelocal/var/security/adm/DCEERS file 543
94. The /opt/dcelocal/var/security/adm/PROCENTR file 543
95. Layout of the /opt/dcelocal/var/security/adm/PROCENTR file 543
96. The /opt/dcelocal/etc/IMPTVAR file 545
97. The /opt/dcelocal/var/security/adm/RACFUNLD file 546
98. Layout of the /opt/dcelocal/var/security/adm/RACFUNLD file 546
99. The /opt/dcelocal/var/security/adm/DCEWORK file 547
100. The /opt/dcelocal/var/security/adm/DCENEW file 547

101. The /opt/dcelocal/etc/EXPTVAR file 548
102. The /opt/dcelocal/var/security/adm/RACFWORK file 550
103. The /opt/dcelocal/var/security/adm/RACFERS file 550
104. The /opt/dcelocal/var/security/adm/RACFNEW file 551
105. The /opt/dcelocal/var/security/adm/ASUIDMAP file 551
106. The Identity Mapping File 551
107. The /opt/dcelocal/var/security/adm/PRNIDMAP file 552
108. The Principal Mapping File 552

Tables

1. Availability of DCE Daemons in OS/390 DCE . . . 17
2. Availability of DCE Administrative Facilities . . . 18
3. OS/390 Administration Commands . . . 30
4. OS/390 DCE User Commands . . . 30
5. Permissions Used by the DCE Daemon . . . 150
6. The dcecp Command Operations for CDS . . . 193
7. The cdscp Program Commands with No Equivalent dcecp Command . . . 195
8. ACL Entry Types Supported by CDS . . . 199
9. The dcecp Program Commands and Required Permissions . . . 201
10. The cdscp Commands and Required Permissions . . . 203
11. Permissions Required To Create Target Objects . . . 236
12. DTS Command Operations . . . 271
13. Settable DTS Object Attributes . . . 272
14. Unsettable DTS Object Attributes . . . 272
15. ACL Entry Types for Principals and Groups . . . 311
16. Security Service Command Operations . . . 328
17. The sec_admin Commands for Reconfiguring Replica Sets . . . 330
18. Attribute Options to Create Principals . . . 334
19. DCE Release-to-Release Authentication Interoperation . . . 338
20. Attribute Options to Create Groups and Organizations . . . 342
21. Attribute Options to Create Accounts . . . 352
22. The keytab create and add Options . . . 358
23. Options to Create Extended Attributes . . . 364
24. Encoding Types . . . 367
25. The registry connect Command Options . . . 377
26. Default Attribute Values of Cross-Cell Authorization Principals and Accounts . . . 378
27. Summary of variable file settings and their resulting RACF command parameters . . . 395
28. Stricter Standard Policies . . . 414
29. Permissions for Registry Objects . . . 430
30. ACL Managers, Valid Permissions, and ACL Entry Types. . . . 442
31. OS/390 DCE Environment Variables . . . 467
32. The Code Set Registry . . . 488
33. Metacharacters and Their Meaning . . . 531
34. Summary of CDS, GDS, and DNS Characteristics . . . 531
35. Maximum Sizes of Directory Service Names . . . 533
36. PROCENTR Data Record Format . . . 544
37. RACFUNLD Data Record Format . . . 547

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM 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, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785, USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Request

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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

The following terms are trademarks of the International Business Machines Corporation in the United States or other countries or both:

AIX	AIX/6000	BookManager
CICS	DB2	DFSMSdss
DFSMS/MVS	DFSORT	eNetwork
ESCON	IBM	IBMLink
IMS	Library Reader	MVS
OpenEdition	OS/2	OS/390
Parallel Sysplex	RACF	Sysplex Timer
System/390	S/370	S/390

Websites in This Book

Any pointers in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this publication or accessed through an IBM Web site that is mentioned in this publication.

About This Book

The purpose of this book is to help system and network administrators understand and administer OS/390 DCE. While many administrative functions can be performed on any node in a distributed environment, this book describes how to perform these tasks on an OS/390 host only.

Who Should Use This Book

This book is intended for system and network administrators who understand the fundamental concepts of a distributed environment. The reader is assumed to be well versed with the OS/390 DCE environment. A knowledge of TCP/IP communications and the UNIX operating system can help administrators use this book more effectively. Administrators who have little or no experience with Distributed Computing Environment (DCE) are advised to read *Distributed Computing Environment: Understanding the Concepts*, GC09-1478, before using this guide.

How to Use This Book

This guide is divided into the following parts:

- Part 1, "DCE Administration Concepts" on page 1 provides a conceptual overview of DCE system administration, introduces the DCE components, and presents the tasks involved in maintaining those components.
- Part 2, "General Administration Aspects of OS/390 DCE" on page 27 contains information on the implementation of DCE on OS/390.
- Part 3, " The DCE Control Program" on page 43 introduces the control program for DCE and provides general information on how to use it.
- Part 4, " DCE Administration Tasks" on page 81 presents the main categories of tasks that the DCE administrator needs to perform.
- Part 5, " DCE Host and Application Administration" on page 97 provides information on tasks for managing host services, host data, and applications.
- Part 6, "DCE Directory Service" on page 157 describes the DCE Directory Service component, specifically, the Cell Directory Service.
- Part 7, "DCE Distributed Time Service" on page 257 focuses on the DCE Distributed Time Service component, and describes the administrative services associated with this component.
- Part 8, "DCE Security Service" on page 293 introduces the DCE Security Service component, and provides an overview of commands and other administrative tools.

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, options, or path names.
Italic	*Italic* words or characters represent values for variables that you must supply.
`Example font`	Examples and information displayed by the system appear in `constant width type style`.

[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices.
< >	Angle brackets enclose the name of a key on the keyboard.
...	Horizontal ellipsis points indicate that you can repeat the preceding item one or more times.
\	A backslash is used as a continuation character when entering commands from the shell that exceed one line (255 characters). If the command exceeds one line, use the backslash character \ as the last nonblank character on the line to be continued, and continue the command on the next line.

This book uses the following keying conventions:

<**Alt-***c*>	The notation <**Alt-***c*> followed by the name of a key indicates a control character sequence.
<**Return**>	The notation <**Return**> refers to the key on your keyboard that is labeled with the word Return or Enter, or with a left arrow.
Entering commands	When instructed to enter a command, type the command name and then press <**Return**>.

Product Name

The product name **OS/390 DCE** refers to the DCE services on OS/390.

Where to Find More Information

It is recommended that this book be used together with *OS/390 DCE: Command Reference* which provides the syntax of the DCE administration commands.

For information about planning, installing, and configuring DCE components, refer to *OS/390 DCE: Planning*, SC28-1582, *OS/390 Program Directory*, and *OS/390 DCE: Configuring and Getting Started*, SC28-1583.

Where necessary, this book refers to information in other books, using shortened versions of the book title. For complete titles and order numbers of the books for all products that are part of OS/390, see the *OS/390 Information Roadmap*, GC28-1727. For complete titles and order numbers of the books for OS/390 DCE, refer to the publications listed in the "Bibliography" on page 591.

The following list gives titles and order numbers for books related to other products:

- :citdocid=icea100.DFSORT™ Application Programming Guide, SC33-4035.
- *The Distributed File System (DFS) for AIX/6000®*, GG24-4255.

For information on how to use the Tool Command Language, see *Tcl and the Tk Toolkit*, John K. Osterhout, (c)1994, Addison—Wesley Publishing Company.

For a list of other DCE and related publications, see the "Bibliography" on page 591, which appears at the end of this book.

Information about DCE administration on other IBM® systems can be found in the administrator's guide for those systems.

Online Books

All the books belonging to the OS/390 DCE library are available as online publications. They are included in the *IBM OS/390 Collection*, SK2T-6700.

All the books in the Online Library are viewable, without change, on these IBM operating platforms: OS/390, VM, OS/2®, DOS, and AIX/6000®. The same book can be viewed on any of these platforms using the IBM BookManager® Library Readers™ for OS/2, Windows, and DOS, or any of the IBM BookManager READ licensed programs for OS/390, VM, OS/2, Windows, DOS, or AIX/6000.

The booklet included with the Online Library provides details on accessing the OS/390 DCE online publications.

Summary of Changes

Summary of Changes
for SC28-1584-05
OS/390® Version 2 Release 7

This book contains information previously presented in the *OS/390 DCE Administration Guide*, SC28-1584-04, which supports OS/390 Version 2 Release 6.

The following summarizes the changes to that information.

New Information

- The base TCP/IP Services component of the eNetwork™ Communications Server for OS/390 can now be used with OS/390 DCE. There are some DCE-specific restrictions.
- If your installation has many DCE clients logging into DCE from OS/390, the performance of these logins can be improved by using a Temporary File System (TFS).
- Interoperability of OS/390 DCE with clients based on OSF DCE 1.2.2 is now possible.
- There are seven new environment variables to be used with a security server.
- There is a new environment variable to be used for debugging.

Changed Information

- What to do when removing a replica that is a security server.
- When the security server is on your local system, it is no longer necessary to store the password in the RACF registry.

This book includes terminology, maintenance, and editorial changes that are not marked. Technical changes or additions to the text and illustrations, however, are indicated by a vertical line to the left of the change.

Summary of Changes
for SC28-1584-04
OS/390 Version 2 Release 6

This book contains information previously presented in the *OS/390 DCE Administration Guide*, SC28-1584-03, which supports OS/390 Version 2 Release 5.

The following summarizes the changes to that information.

New Information

- There are enhancements to DCE that now permit workload balancing among servers in a Parallel Sysplex® environment using Coupling Facility and Workload Manager.
- DCE administrators can now use the Control Task to monitor utilization of the Hierarchical File System (HFS) by DCE.
- There are a number of new environment variables. These are for:
  - Balancing server workloads in a Parallel Sysplex environment
  - Monitoring the use of HFS by DCE
  - Using the audit trail file

- Configuring a cell
- Setting up passwords
- Setting up the security server
- Using the IDL compiler

Changed Information

- As part of the name change of OpenEdition® to OS/390 UNIX System Services, occurrences of OpenEdition have been changed to OS/390 UNIX System Services or its abbreviated name, OS/390 UNIX. OpenEdition may continue to appear in messages, panel text, and other code locations.

Summary of Changes
for SC28-1584-03
OS/390 Version 2 Release 5

This book contains information previously presented in the *OS/390 DCE Administration Guide*, SC28-1584-02, which supports OS/390 Version 2 Release 4.

The following summarizes the changes to that information.

New Information

- Several new environment variables have been added to support the use of Kerberos V5R1 for security.

Summary of Changes
for SC28-1584-02
OS/390 Version 2 Release 4

This book contains information previously presented in the *OS/390 DCE Administration Guide*, SC28-1584-01, which supports OS/390 Version 1 Release 2.

The following summarizes the changes to that information.

New Information

- Remote Procedure Call (RPC) Internationalization support
- OS/390 Cell Directory Server
- Global Directory Agent (GDA) support, including use of an external naming service that supports the Lightweight Directory Access Protocol (LDAP).
- Hardware Cryptography support

Summary of Changes
for SC28-1584-01
OS/390 Version 1 Release 2

This book contains information previously presented in the *OS/390 DCE Administration Guide*, SC28-1584-00, which supports OS/390 Version 1 Release 1.

The following summarizes the changes to that information.

Changed Information

- Several changes as a result of support for National Languages. This impact is felt mostly in:
  - The table of environment variables
  - Use of code page IBM-1047 for **dcecp** scripts

Part 1. DCE Administration Concepts

Chapter 1. Introduction to DCE System Administration . . . 3
Clients and Servers . . . 3
Remote Procedure Call . . . 4
Cells . . . 4
The DCE Namespace . . . 5
Filespaces . . . 5
Principals . . . 5
Access Control Lists . . . 6
Caching . . . 6
Replication . . . 6

Chapter 2. Overview of DCE Components . 7
Description of DCE Components . . . 8
DCE Remote Procedure Call (RPC) . . . 8
DCE Directory Service . . . 9
DCE Distributed Time Service (DTS) . . . 12
DCE Security Service . . . 14
How the DCE Components Work Together . . 15
Remote Procedure Call . . . 16
Directory Service . . . 16
Distributed Time Service . . . 16
Security Service . . . 17
What Makes Up DCE System Administration? .17
Summary of DCE Daemons and Administrative Facilities . . . 17

Chapter 3. Overview of DCE Maintenance . 19
RPC Maintenance Tasks . . . 19
Maintaining NSI Entries . . . 19
Maintaining the Endpoint Map . . . 19
Controlling Access to the Endpoint Map . . 19
| Restricting Network Interfaces Used by
| DCE . . . 20
Cell Directory Service Maintenance Tasks . . . 20
Monitoring the Cell Directory Service . . . 20
Managing the Cell Directory Service . . . 21
Cell Directory Service Security and Access Control . . . 21
Distributed Time Service Maintenance Tasks . 22
Managing the Distributed Time Service . . . 22
Modifying the System Time . . . 23
DCE Security Service Maintenance Tasks . . . 23
Reconfiguring the Registry . . . 24
Improving Registry Performance in Large Cells . . . 24
| Improving dce_login Performance . . . 25

Chapter 1. Introduction to DCE System Administration

The *OS/390 DCE: Introduction* introduced you to the Distributed Computing Environment (DCE), describing the major components of its services. This chapter gives an overview of DCE from the perspective of the system or network administrator.

DCE is a set of services that together make up a high-level, coherent environment for developing and running distributed applications. These services provide a set of tools that support DCE management tasks. DCE applies techniques you may have learned from working with applications for single machines or other distributed systems. These techniques help you to manage DCE without having to know about system internals. You can start with a configuration 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.

You need to understand the following concepts when performing DCE system administration:

- **Clients** and **servers**, which make and respond to, requests for a service
- **Remote Procedure Calls** (RPCs) for client-to-server communication
- **Cells**, which are groups of users, servers, and machines that share security, administrative, and naming boundaries
- A single **namespace** that is used by client applications to identify, locate, and manage **objects**, including users, machines, servers, groups of servers, and directories
- A single **filespace** that is used for data sharing among users and machines with proper authorization
- **Principals**, which are entities—users, servers, and machines—that are involved in secure communications with other entities
- **Access Control Lists** (ACLs), which control access to objects
- **Caching**, the technique of using a local copy of information to avoid looking up the centrally stored information each time it is needed
- **Replication**, the process by which copies of information are created and kept consistent.

The following sections describe these concepts in more detail.

Clients and Servers

DCE is based on the client/server model. A **client** is a machine or a process that makes use of a server's specialized service during the course of its own work. A **server** is a machine or a process that provides a specialized service to other machines or processes. Distributed applications consist of a client side that initiates a request for service, and a server side that receives and runs 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 DCE component. In the same manner, an application server and a client can be running on the same host system.

The client and server applications are totally network-transparent. They can be running anywhere within the cell (or even in a foreign cell) but the client doesn't need to know anything about how to get to the server.

Figure 1 on page 4 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.

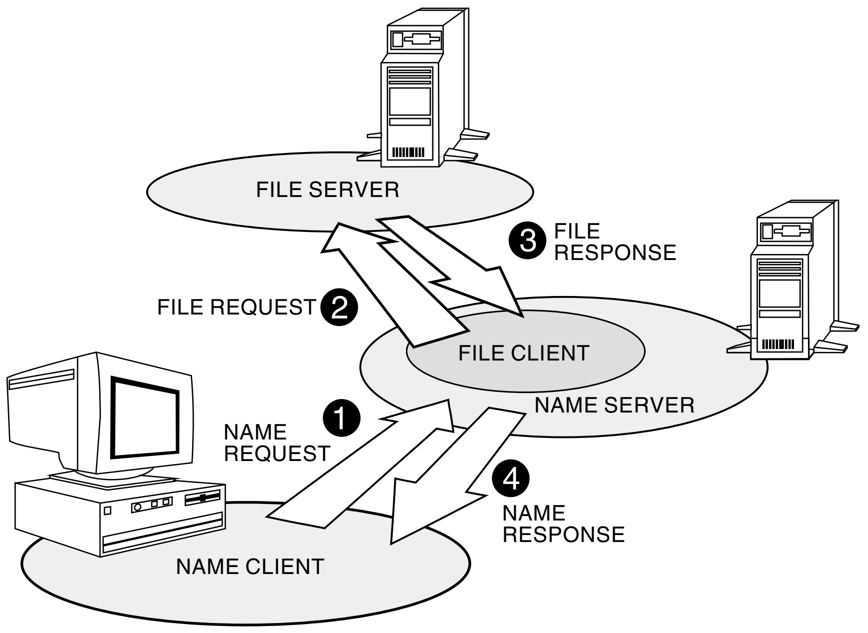


Figure 1. Interaction of Clients and Servers

Remote Procedure Call

A **remote procedure call** (**RPC**) is a synchronous request and response between a local calling program and a remote procedure. A remote procedure call begins with a request from a local calling program to use a remote procedure and is completed when the calling program receives all the results from the procedure (or an error status or exception).

Cells

The **cell** is the basic unit of operation 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 be located in different buildings, different cities, or even different countries, provided they are adequately connected. A cell can range in size from one machine to several thousand, depending on the size of an 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 have the same security requirements, 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 illustrated 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 directory and security servers use.
- Pathnames of DCE objects managed by DCE services can be expressed relative to the cell where the objects reside.
- The **DCE Distributed Time Service** (DTS) has both local and global servers. Local servers operate within the Local Area Network (LAN) (depending on the LAN Profile, local servers can also operate within the Wide Area Network). Global time servers provide time services anywhere within the cell.

The DCE Namespace

The DCE namespace is the hierarchical set of names of DCE objects. The top levels of the hierarchy are managed by the DCE Directory Service. Some DCE services like the DCE Security Service manage their own portions of the DCE namespace. Each DCE object in the namespace consists of a name with associated attributes (pieces of information) that can be used to locate it. These objects include resources such as machines or applications.

The DCE namespace contains cell namespaces and global 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 DCE Directory Service. Conversely, the global namespace, as seen from a local DCE cell, contains objects that are registered outside the cell, such as the names of other cells. The **Global Directory Service** (**GDS**) component of the DCE Directory Service may be used to manage the global namespace. Alternatively, a non-DCE service such as the **Domain Name System** (**DNS**), or a server that supports the **Lightweight Directory Access Protocol (LDAP)**, may also be used to manage the global namespace. (The term “LDAP server” is used in this publication to refer to a server that supports LDAP.)

Note: The Global Directory Service is not available in OS/390 DCE.

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

Filespaces

Part of the DCE cell namespace is the filespace, which consists of files and directories that can be physically stored on many different machines but are available to users on every machine, as long as they 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, it appears to users that there is a single filespace.

Principals

A DCE **principal** is the identity that is authenticated by the DCE Security Service. When you do a DCE login, you use your principal name. Principals can be organized into groups of principals and into organizations that contain groups of principals. Information associated with a principal includes the principal's name and group memberships. 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 another cell to participate securely within your cell.

Access Control Lists

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

- Principals, groups of principals, and organizations managed by the DCE Security Service
- Files and file system directories managed by the DCE Distributed File Service (DFS)

  Note: DFS has its own set of publications. Consult those manuals for more information.
- DTS servers
- CDS directories and entries
- CDS clients and servers that 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.

ACLs are similar to the UNIX** system's file-protection model. Whereas UNIX 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. DCE provides the **dcecp acl** command, which administers ACLs on all DCE objects.

Caching

Information acquired over the network (for example, using 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 DCE namespace and the DCE 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 DCE Security Service, CDS, GDS, and DFS all provide replication facilities that are customized for their particular application.

Note: OS/390 DCE does not provide GDS. Therefore, replication as described in this book applies only to these services that reside on other host systems. The CDS Server *is* part of OS/390 DCE Base Services.

Although the OS/390 Security Server is not part of OS/390 DCE Base Services, it is available as an optional feature of OS/390. The management interfaces (**dcecp**, Registry Editor, and ACL Editor) are part of the OS/390 DCE Base Services.

DFS is available as a base element of OS/390.

Chapter 2. Overview of DCE Components

This chapter provides an overview of the components you will use in administering DCE. It also describes how these components relate to each other, and explains some key terms used in DCE administration. This chapter ends with Table 1 on page 17 and Table 2 on page 18, which summarize the daemons and administrative facilities of OS/390 DCE. *Distributed Computing Environment: Understanding the Concepts* contains a discussion of all the DCE components and includes information about functions that application developers use.

Figure 2 shows the different OS/390 DCE components and how they fit together. OS/390 DCE resides between the applications shown at the top of the illustration and the operating system and transport services at the bottom. Except for the DCE Threads Service and the OS/390 DCE Application Support, the shaded boxes show DCE components that involve system administration tasks discussed in this book.

Note: Although the OS/390 Security Server is not part of OS/390 DCE Base Services, it *is* available as an optional feature of OS/390. The management interfaces (**dcecp**, Registry Editor, and ACL Editor) are part of the OS/390 DCE Base Services.

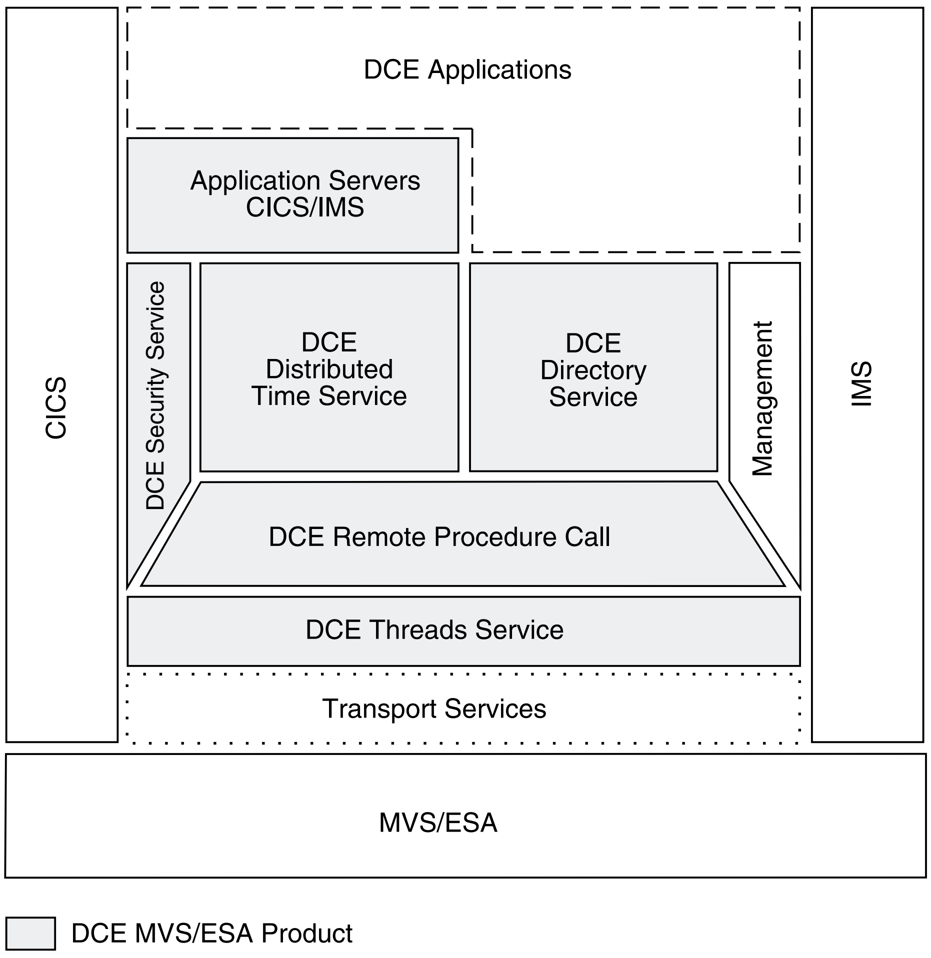


Figure 2. OS/390 DCE

Description of DCE Components

While each of the DCE components serves a separate function, they are interrelated. This section describes the DCE components that pertain to system administration.

Although **Threads** is a fundamental component of DCE, it is not discussed here or in the other parts of the book. There are no administrative tasks associated with this component.

The Management block shown in Figure 2 on page 7 represents the administrative tools that assist you in managing DCE. They are described in the appropriate sections of this book.

DCE Remote Procedure Call (RPC)

DCE Remote Procedure Call (RPC) is the primary method for client-to-server communication in DCE. DCE RPC allows a program to call a procedure on a remote machine as if it were a local procedure call. To the application programmer, a remote call looks almost like a local call, but there are several RPC components that contribute to this facility including the Interface Definition Language (IDL), a Universal Unique Identifier (UUID) generator, and the RPC Runtime Library, which supports two RPC protocol implementations. One protocol operates over connection-oriented transports such as the Transmission Control Protocol (TCP/IP), and the other protocol operates over connectionless transports such as the User Datagram Protocol/Internet Protocol (UDP/IP).

An end user does not see RPC at all, and the minimal amount of administration in RPC (such as advertising an application server in the DCE Directory Service), can usually be handled by the application program on the server. Because many of the DCE components are themselves client-server applications, they use RPC for distributed communications.

In addition to IDL and its compiler, the UUID generator, and the Runtime Library, other parts of DCE that contribute to RPC are:

- **Authenticated RPC**

  The DCE Security Service provides RPC secure communications.

- **Name Service Independent (NSI) API**

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

- **The DCE host daemon (dced)**

  The daemon is a program that runs on every DCE machine. It includes, among other things, an RPC-specific name server called the endpoint mapper service. This service manages a database that maps RPC servers to the transport endpoints that the server is listening for requests on.

- **The DCE Control Program (dcecp)**

  The DCE Control Program is a tool for administering DCE.

Each RPC-based server must register its **addressing** (or **binding**) information so that its clients can find it. The addressing information typically consists of two parts. The first part is the address of the machine on which the server runs. This information is stored in CDS, and gives enough information for a client to find the DCE daemon on the server's machine, because **dced** has a well-known endpoint. The second part of the server's address is the server's endpoint. (For the Internet Protocol or IP, an endpoint is a port.) The server gets a dynamically assigned endpoint when it starts up. The server must register its endpoint with **dced**; this is usually done during server initialization. The **dced** maintains RPC server information in a database, the **endpoint map**. The client contacts **dced** (which listens on a well-known endpoint) on the server's machine and requests the server's endpoint. It can then locate the server.

DCE Directory Service

The DCE Directory Service provides directory service at the cell and global levels. It allows both users and applications to store, retrieve, and manage information about objects such as computers, printers, users, and files. Because the DCE Directory Service facilitates the use of common naming conventions within a common namespace, users and applications are not restricted by physical location, brand of host system, or method of naming on a host system. Using common naming conventions allows sharing of information based on names, rather than on location.

The DCE Directory Service stores an object and its attributes. An **attribute** is a piece of information associated with an object. Attributes can describe an object's class, network address, or other values. Therefore, an object's name does not need to change if it moves from one node to another. You can also search for a name given one or more of an object's attributes if the object is stored in the GDS part of the namespace.

Note: Because OS/390 DCE does not support the Global Directory Service (GDS), administration of this DCE Directory component must be performed on the host that offers this service.

Cell Directory Service: CDS is the component that looks up and manages names within a cell. Client applications send their requests through a **CDS clerk** process, and, if the data is not in a cache, it is passed to one or more servers to be handled. The CDS clerk caches information obtained from the server for subsequent lookups.

The CDS server stores names and other CDS information in a database called a **clearinghouse**. The clearinghouse contains **replicas**, which are physical instances of CDS directories containing names.

Another process, the **CDS Advertiser**, is responsible for sending and receiving advertisements of the presence of CDS servers on DCE machines. It also creates the cache that is used by the CDS Clerk.

There is a fundamental difference between the OSF** DCE and the OS/390 DCE implementations of the CDS clerk. In the OSF DCE implementation, on each machine, one CDS clerk is started by the CDS Advertiser for each principal that makes CDS requests on the machine. Therefore, several CDS clerk processes can be running on a machine at any one time. In OS/390 DCE, only one CDS clerk runs on each machine and it processes the CDS requests from all clients on the machine.

The CDS administration tasks you perform are:

- Configuring and replicating the CDS namespace
- Managing CDS servers
- Managing access control on CDS directories and entries.

After CDS is installed and configured, only occasional intervention for system administration is required. OS/390 DCE provides the DCE control program (**dcecp**) to support most, but not all of the CDS system administration tasks. (Some functions of CDS still use the earlier CDS control program (**cdscp**).)

The DCE control program is an interactive, command-line interface you can use to configure the CDS namespace and perform maintenance tasks such as monitoring servers or creating a directory. With **dcecp** commands, you can also display the structure and content of the CDS namespace.

Note: Some administration tasks such as creating and deleting clearinghouses can only be performed on the machine where the CDS server is running.

Global Directory Servers (LDAP and X.500): Global directory servers support the global naming environment outside of cells. Examples of global directory servers are:

- The Global Directory Service (GDS), which conforms to the X.500 directory service standard
- The Lightweight Directory Access Protocol (LDAP) Server

Note: GDS is not available in OS/390 DCE.

Global directory servers are somewhat independent of the DCE cell; for example, they do not use DCE RPC for interprocess communications. They maintain a directory that is used by DCE to store information about DCE cells. This directory can also be used as a general-purpose directory service.

The global directory server directory is a distributed database. Each Directory System Agent (DSA), which is the server side of GDS, stores a different part of the database. A DSA can have copies of the information of other DSAs to increase availability and reduce response times. The original information is called a **master** and the copy is called a **shadow**. Every object in a DSA is either a master or a shadow. When an update occurs, usually the master object is changed. You can create jobs that periodically update shadows. You can periodically update the shadows for any modifications done on the master.

The client side of a GDS configuration is known as a **Directory User Agent** (DUA). The client side of an LDAP Server is called the **LDAP Client Runtime Library**.

The Global Directory Agent: The third component of the DCE Directory Service is an independent process called the **Global Directory Agent** (**GDA**). Not previously supported in OS/390 DCE, GDA now provides these functions:

- Name resolution for untyped names using DNS
- Name resolution for typed names using an LDAP server
- On non-OS/390 platforms, name resolution for typed names using an X.500 server

Notes:

1. GDA in OS/390 does *not* support name resolution using an X.500 server.
2. See the *OS/390 DCE: Application Development Reference* for application programming interfaces (APIs) that work with an LDAP server.

When CDS receives a directory request, it determines whether it can find the object in its own cell or whether it needs to contact another server or service to help it find the object. If the object is stored in another (**foreign**) cell, the CDS server of the other cell must be called to resolve the name. To contact CDS in the foreign cell, the local CDS must know the location of the foreign cell. CDS contacts the GDA to assist CDS in finding the location of the other cell.

A GDA is only needed in a cell if communication with directory services in other cells is required. It can reside on any host within the cell.

The administrative tasks for GDA consist of:

- Starting and stopping the GDA process
- Deciding how many GDA processes need to run in the cell.

It is also the DCE administrator's responsibility to perform the necessary GDS, LDAP server, or DNS tasks required to add an entry for the CDS machine within the cell that receives the requests from foreign cells.

Cell names and attributes can be registered in GDS, the LDAP server, or another global name service, the **Domain Name System** (DNS). Although DNS is not part of the DCE offering, it is supported by the DCE Directory Service and requires some administration. DNS is part of the TCP/IP product.

On OS/390, the GDA determines whether the foreign cell object is in the LDAP server or DNS, depending on the format of the cell name. The cell name is contained in the prefix of the CDS object name.

X.500 names are **typed**, consisting of a type and a value separated by an "=" (equal sign). If the GDA encounters a name such as **/C=US/O=ABCcompany**, it knows that the entry belongs in an LDAP server or an X.500 server.

DNS names are **untyped**, consisting of one or more values separated by a period. If the GDA encounters a name such as **mycell.abcCompany.com**, it knows that the entry is found in DNS.

Typed and untyped names are structured according to X.500 and DNS naming conventions.

Figure 3 and Figure 4 show a very simplified representation of how the GDA helps CDS resolve a name that is stored in an LDAP server.

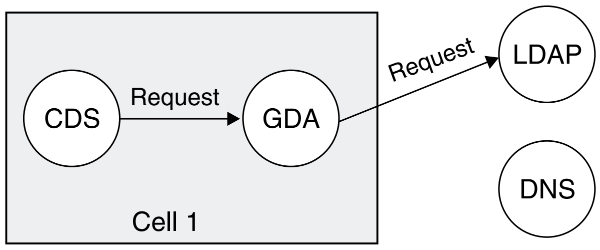


Figure 3. CDS Requests a Name in Another Cell

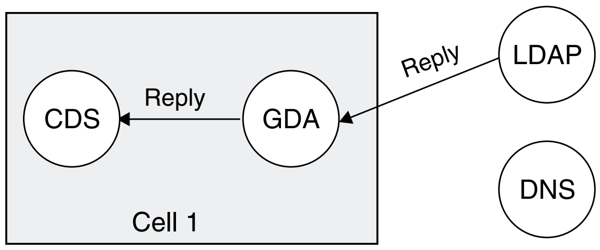


Figure 4. Location Information Is Returned to CDS

With this information, CDS can now directly access the other cell, as shown in Figure 5 on page 12.

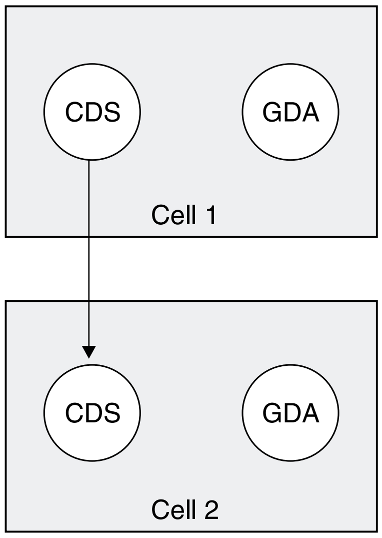


Figure 5. CDS Contacts Another Cell

To use both the local and global naming services that the DCE Directory Service provides, a cell must contain at least one CDS server and at least one GDA. A cell can use only CDS for local directory service and not use the GDA, but this cell cannot refer to objects in other cells.

For more information on CDS and GDA, refer to the following sections:

- Chapter 18, “Introduction to the DCE Directory Service” on page 161 and Chapter 19, “Cell Directory Service Concepts” on page 173, which provide detailed discussion of CDS and the DCE Directory Service.
- Chapter 29, “Managing Intercell Naming” on page 251, which provides details on how the GDA works.

DCE Distributed Time Service (DTS)

DTS synchronizes the clocks in networked computer systems. It checks time synchronization and adjusts clocks when the clock error exceeds a certain acceptable range, which you can set. Client applications can also use the application programming interface provided by DTS to manipulate timestamps.

Figure 6 on page 13 shows the DTS relationships.

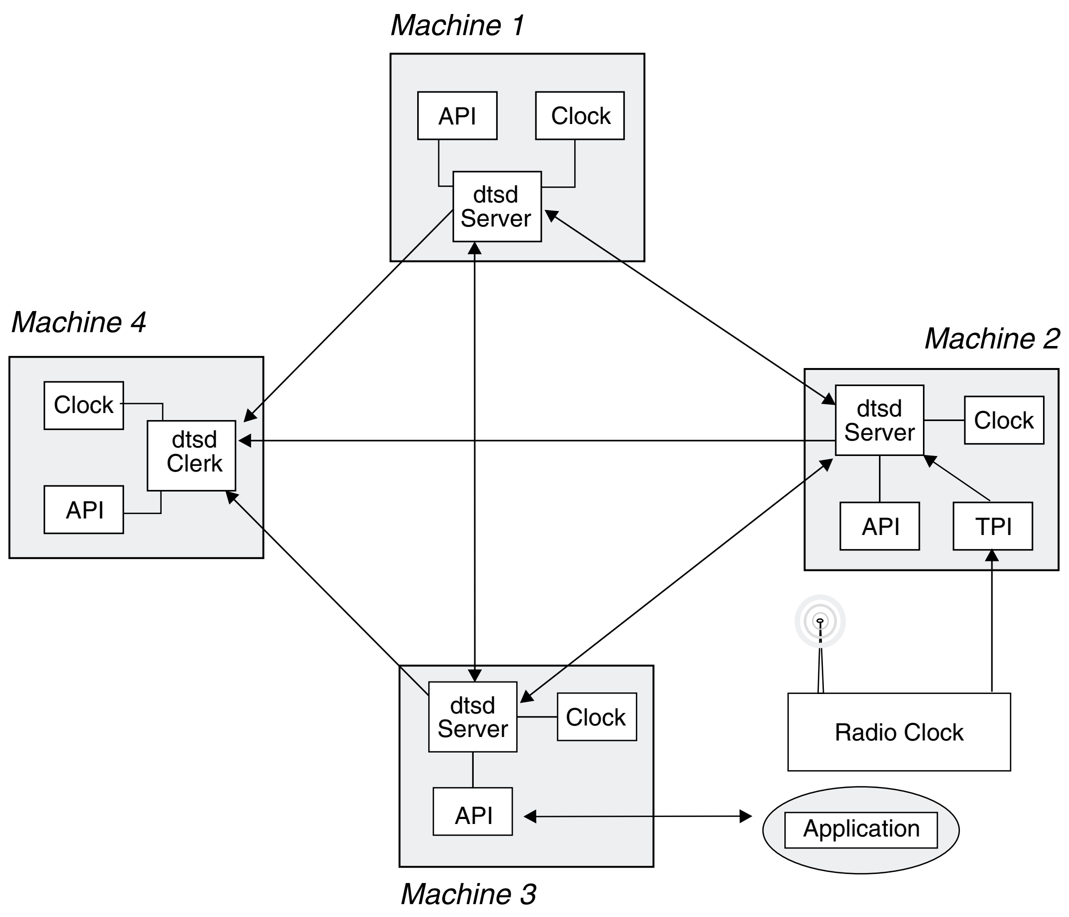


Figure 6. DTS Relationships

A DTS daemon runs on each DCE machine. Most of the DTS daemons are configured as clerks. DTS clerks are responsible for receiving time values and adjusting the system clock accordingly. Some DTS daemons are configured as servers. DTS servers are responsible for synchronizing times among each other, as well as performing DTS clerk tasks.

DTS has a **Time Provider Interface** (**TPI**) that allows a server to import time values from outside time sources, such as radio, telephone, or satellite.

Note: TPI is supported in OS/390 DCE, but OS/390 UNIX System Services provides only a null time-provider program.

You use the DCE control program (**dcecp**) **clock**, **dts** and **utc** commands to perform most, but not all DTS configuration and management tasks. (Some functions of DTS still use the earlier DTS control program (**dtscp**).) The DTS synchronization functions run as background processes, and after DTS is installed and configured, the service does not require much intervention from the system administrator. You may need to adjust DTS configurations to meet varying conditions within your DCE cell. DTS administration tasks include the following:

- Registering DTS servers as objects in the CDS namespace
- Configuring additional DTS servers
- Setting the inaccuracy limit that forces synchronization, to bring the inaccuracy back to an acceptable level.

Chapter 30, “Introduction to the DCE Distributed Time Service” on page 259 explains how DTS works. The chapter includes discussions of DTS time representation and basic time and clock concepts.

DCE Security Service

The DCE Security Service enables clients and servers to prove their identities to each other. It offers integrity and privacy of communications and supports controlled access to resources. It acts on behalf of principals. In DCE, principals are represented as entries in the DCE Security Service's database, called the **registry**. These entries include users, servers, and machines.

The DCE Security Service provides tools to help you administer Security on both the local machine and the cell.

Managing the local machine includes running commands that add, delete, or list the key table entries used by non-interactive users, such as machines and server processes.

Cell administration includes managing the security server and creating and maintaining information kept in the registry using a Registry Editor.

Note: Although the OS/390 Security Server is not part of OS/390 DCE Base Services, it *is* available as an optional feature of OS/390 and can run on any OS/390 host in the cell. The management interfaces (**dcecp**, Registry Editor, and ACL Editor) are part of the OS/390 DCE Base Services. The IBM OS/390 Security Server *is* available as an optional feature of OS/390.

The registry contains information on principals, groups, organizations, accounts, and administrative policies. Each cell has one registry database. It can also have replicas known as **slaves**.

You can use **dcecp** to set up accounts for foreign cells in your cell's registry, indicating that you trust the Authentication Service in the foreign cell to correctly authenticate its users.

The DCE Security Service consists of several cooperating services and facilities. One of these services is the **Registry Service**, which helps you manage user, group, and account information. In addition to the Registry Service, the DCE Security Service includes the following services and facilities that require very little or no system administration:

- The **Authentication Service**.

  Provides trustworthy identification of principals involved in network operations. A principal gains access to DCE by means of an account, which consists, in part, of the principal name and a secret key (password) that the principal shares with the Authentication Service.

- The **Privilege Service**.

  Certifies a principal's identity and group membership. A principal's identity and group membership, also known as **privilege attributes** can be used by an Access Control List (ACL) to determine a principal's access permissions to objects. The Privilege Service provides the privilege attributes that can be used to determine if a principal has the right to do what it wants to do.

- The **DCE ACL Facility**.

  Determines a principal's access to an object by comparing entries in the object's ACL to the identity and group membership of the principal. The **dcecp acl** command is the administrative tool used to create, change, and delete ACL entries. Other DCE components use the ACL model provided by the DCE Security Service through their individual ACL Managers.

- The **DCE Login Facility**.

  Initializes a user's DCE Security environment. It also authenticates the user to the Security Service by means of the user's password, thereby establishing an authenticated network identity.

See Part 8, “DCE Security Service” on page 293 for detailed information about administration of the DCE Security Service.

RACF Interoperability and Single Sign-on: OS/390 DCE also provides interoperability between Resource Access Control Facility (RACF®) on OS/390 and OS/390 DCE. This security interoperability allows a DCE client to access a DCE-enabled server on an OS/390 UNIX System Services system and allows the DCE server to acquire corresponding local security credentials for a DCE client to access OS/390 resources. The interoperability function allows:

- Appropriately authorized DCE servers to acquire corresponding OS/390 security credentials for the DCE client and to use the DCE client's corresponding OS/390 user ID for access to RACF-authorized resources.
- An OS/390 user to be transparently logged in to DCE when necessary, without prompting for a DCE user ID or password. This ability is called **single sign-on**. With this feature, a user authenticates to OS/390 and can run a DCE program without reauthenticating to DCE.

OS/390 DCE also provides RACF interoperability administration utilities. These incorporate into RACF the information that associates an OS/390 user ID with a DCE principal's identifying information and the DCE principal's UUID with the corresponding OS/390 user ID. This is called **cross-linking information** and is what allows interoperability and single sign-on to work.

The cross-linking information must be set up before interoperability functions can be used. To do this, DCE provides two utilities, **mvsimpt** and **mvsexpt**, for creating the initial cross-linking between the RACF database and DCE registry. This cross-linking can be done from either the RACF database or the DCE registry, but **mvsimpt** and **mvsexpt** must be run from the OS/390 system where the RACF database resides whose users are to be cross linked.

OS/390 also provides application programming interfaces (APIs) so that you can write your own server programs to take advantage of RACF interoperability.

See Chapter 41, “RACF Interoperability and Single Sign-on” on page 389 for more detailed information.

How the DCE Components Work Together

Although DCE consists of distinct components, these components are integrated and interrelated. This section summarizes the relationships between components that have associated system administration tasks.

Most DCE components rely on RPC, the DCE Directory Service, DTS, and the DCE Security Service. The interaction is often reciprocal. For example, RPC uses the DCE Security Service's Authentication Service to get **tickets** and keys, and the Privilege Service to securely associate clients with their identities. The DCE Security Service, in turn, uses RPC for its communications.

The CDS component of the DCE Directory Service, DTS, and the DCE Security Service, along with RPC are the components that every DCE cell requires.

The GDA and GDS components of the DCE Directory Service and DFS are not required for a minimum DCE configuration. If these services are part of your cell, they rely on some or all of the services mentioned in the previous paragraph.

Remote Procedure Call

An RPC server can store information about itself in CDS. An RPC client can look up location information about RPC servers in CDS when it wants to make a call to a particular server. CDS returns information that RPC libraries interpret as binding information and turn into a binding handle. The **binding handle** identifies the RPC server so the RPC client can make its RPC call.

RPC uses the DCE Security Service for **authenticated RPC**, the process by which RPC clients and servers are identified to one another, and by which privacy and integrity of communications are maintained. To use authenticated RPC, clients and servers must be running as principals, have accounts, and have performed login operations.

Each RPC program is likely to require some administration of CDS namespace entries and directories, as well as some server-specific file administration.

Directory Service

CDS servers and CDS clients use RPC and the DCE Security Service's Authentication Service to communicate with each another. CDS can also store information about the location of the RPC servers and interfaces that the RPC servers support.

CDS uses the ACL model provided by the DCE Security Service to ensure authorized access to directory data in CDS. To authenticate CDS interactions, CDS uses authenticated RPC provided through the DCE Security Service.

When you create entries in the CDS namespace, a timestamp accompanies the entry. The timestamp is used for propagation to replicas and the expiration of temporary entries. CDS relies on DTS to maintain clock synchronization in the network so that the timestamps are accurate.

CDS uses GDS to find names outside of a cell by means of the GDA. Other DCE components interact with CDS for directory service (global and local), but only CDS and application programs access GDS directly.

Unlike CDS and other DCE components, GDS does not use RPC for its communications. GDS has its own security implementation and does not depend on the DCE Security Service. GDS conforms to the international standard X.500 protocols.

Distributed Time Service

Like CDS, DTS uses RPC to handle communications between DTS servers and DTS clerks.

DTS registers the servers that synchronize system clocks in the network with CDS and also uses CDS to find DTS servers and their locations.

To authenticate DTS interactions, DTS uses authenticated RPC provided through the DCE Security Service. DTS also uses DCE ACLs to control which users can run certain **dcecp** commands. The permissions required to run these commands are discussed in *OS/390 DCE: Command Reference*.

Security Service

The DCE Security Service uses RPC for its communications. The DCE Security Service registers the location of its Security servers (**secd** daemons) with CDS. Other servers in the network can use CDS to locate the Security servers. You manage the namespace entries using the CDS Control Program.

The DCE Security Service relies on DTS to maintain synchronized clocks so that passwords and tickets (used for obtaining network services) are properly time stamped and their expiration is enforced.

The DCE Security Service provides an ACL model for controlling access to objects that are managed by the DCE services. Based on this ACL model, objects and the ACLs on objects are controlled and managed by the DCE service that owns the object. You can use **dcecp** to manage access to principals, groups, and organizations that are registered in the CDS namespace.

RACF Interoperability and Single Sign-on: The RACF interoperability and single sign-on functions of the Security Service make use of the other DCE services in the same way as the rest of the Security functions. Specifically, for OS/390 users on a system that is protected by RACF, single sign-on allows a user who has already been authenticated to RACF to run a DCE program without reauthenticating to DCE. This is done using RPC and is transparent to the user.

What Makes Up DCE System Administration?

DCE system administration can be divided into several parts. You perform some administration tasks only once, while others are part of your daily routine.

The parts of system administration that get you started and enable you to begin using DCE are planning, installing, configuring, and starting up DCE.

OS/390 DCE: Planning contains information about planning for the implementation of OS/390 DCE in your organization.

OS/390 Program Directory provides information about installing the DCE source tape. *OS/390 DCE: Configuring and Getting Started* provides information about configuring and starting up DCE.

Ongoing or maintenance tasks consist of reconfiguring parts of DCE, monitoring DCE components, and performing routine management. These tasks are described in detail in this book.

Summary of DCE Daemons and Administrative Facilities

Table 1 and Table 2 summarize the DCE daemons and administrative facilities, and whether they are available from OS/390 DCE.

Table 1 (Page 1 of 2). Availability of DCE Daemons in OS/390 DCE

Daemon	**Available in OS/390 DCE?**	**Role**
DCE daemon	Yes	Provides services for the local host, and is the server used by remote applications to access these host services.
Security server daemon	No, but available in the IBM OS/390 Security Server.	Provides support for authentication and controlled access to resources.
Security Client daemon	Supported, but functions replaced by DCE daemon.	Ensures that credentials of the machine's principal are up to date.

Table 1 (Page 2 of 2). Availability of DCE Daemons in OS/390 DCE

Daemon	Available in OS/390 DCE?	Role
Audit daemon	Yes	Performs the logging of audit records based on specified criteria.
RPC daemon	Supported, but functions replaced by DCE daemon.	Provides endpoint map service for the host system.
CDS daemon	Yes	Provides directory services to DCE applications in the cell.
CDS Advertiser daemon	Yes	Sends and receives advertisements on the availability of CDS servers.
CDS Clerk daemon	Yes	Acts as the intermediary between the CDS client and the CDS server.
DTS daemon	Yes	Ensures that the time on the host is synchronized with the other hosts in the cell.
GDA daemon	Yes	Locates other cells in a multi-cell environment.
DTS Null Time Provider daemon	Yes	Obtains the time from the host system's hardware clock and gives it to DTS, if it runs as a DTS server.

Table 2. Availability of DCE Administrative Facilities

Facility	Available in OS/390 DCE?	Used to
DCE Control Program	Yes	Perform administration tasks. This program replaces most of the functions of the other programs in this table.
ACL Editor	Yes	Edit the Access Control List of DCE objects.
Registry Editor	Yes	Maintain the Security Registry.
CDS Control Program	Yes	Perform CDS administration tasks.
DTS Control Program	Yes	Perform DTS administration tasks.
RPC Control Program	Yes	Perform RPC administration tasks.
sec_admin Program	Yes	Perform Registry replica administration tasks.
mvsexpt	Yes	Perform DCE-RACF cross-linking administration tasks.
mvsimpt	Yes	Perform DCE-RACF cross-linking administration tasks.
ldap_addcell	Yes	Register cell information in a server that supports LDAP.
mkdceregister	Yes	Generate cell information to be stored in DNS.

Chapter 3. Overview of DCE Maintenance

When the tasks required for planning, installing, and configuring DCE have been performed on your system, you can go on to perform the tasks required for maintaining the system. The initial tasks (planning, installing, and configuring) are performed infrequently, possibly only once. Maintenance tasks, however, are performed on a more regular basis.

Maintenance of a distributed system includes the following elements:

- Performance tuning
- Configuration control
- Security and access control.

This chapter summarizes some of the primary DCE system administration tasks that apply to the individual components of DCE. DCE component tasks are documented in detail in this book.

RPC Maintenance Tasks

Although the DCE Remote Procedure Call is an application programmer's tool, administrative tasks are associated with it. The administrative tasks related to RPC fall into three categories:

- Maintaining NSI entries in the CDS namespace.
- Maintaining the endpoint map database on the local host system.
- Controlling access to the endpoint map.

Maintaining NSI Entries

The RPC Name Service creates entries in the CDS namespace to store binding information of servers running on any host system in DCE. These entries are accessed by servers and clients through the **Name Service Interface** (NSI).

NSI entries consist of server, group, and profile entries. These entries can be created and maintained using the RPC control program.

Maintaining the Endpoint Map

Using the RPC control program, you can:

- Manually register the binding of application servers.
- Display the contents of the endpoint map.
- Monitor and rebuild the endpoint map.

Controlling Access to the Endpoint Map

The endpoint map is an important database that must be protected from unauthorized access. In OS/390 DCE, access to the local endpoint map is controlled by manipulating the **Access Control List** (ACL) **/.:/hosts/***host_name***/config/epmap**.

Restricting Network Interfaces Used by DCE

By default, DCE uses all network interfaces that are defined to OS/390 UNIX System Services. An RPC server gets the list of available network interfaces from the RPC runtime library and uses this information to create the binding information, which is stored in the CDS directory entries and the DCE endpoint map. RPC clients then use this information to contact the RPC server.

The DCE administrator can restrict the interfaces used by DCE in two ways: through the **/opt/dcelocal/etc/rpc_interfaces** file and through the RPC_UNSUPPORTED_NETADDRS and RPC_UNSUPPORTED_NETIFS environment variables.

The **/opt/dcelocal/etc/rpc_interfaces** file is a text file containing one line for each interface to be used by the RPC runtime library. The interface can be specified by either its IP address (for example, 9.130.79.48) or its name (for example, LAN1). If this file does not exist, then all available network interfaces are used by DCE. Otherwise, only interfaces which are specified in the file are used. This file effects all RPC servers running on the local OS/390 system.

The RPC_UNSUPPORTED_NETADDRS and RPC_UNSUPPORTED_NETIFS environment variables can be used to restrict network interfaces on a per-server basis. Network interfaces specified by these environment variables will not be used by DCE even if they are specified in the **/opt/dcelocal/etc/rpc_interfaces** file.

Cell Directory Service Maintenance Tasks

CDS components, including clerks, servers, and clearinghouses, are largely self-regulating. Except for routine monitoring, CDS requires little intervention from you. 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 Chapter 7, "DCE Control Program Introduction" on page 45 to create and manage the components of a CDS namespace.

If you have a large organization, you can improve efficiency by having one system administrator responsible for CDS servers and another 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.

Monitoring the Cell Directory Service

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.

Managing the Cell Directory Service

CDS management tasks fall into the following two categories:

- Managing the namespace:
  – Overseeing the creation of new directories and assigning names to them according to a standard. This may also involve enforcing established guidelines in assigning 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.)
  – Determining default access control policy.
  – Setting and enforcing the established access control policy for directories and entries.
  – Determining where and when new replicas of a directory are necessary.
  – Creating 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 alternative name for an object entry.
  – Resolving problems involving multiple CDS servers.
- Managing CDS servers
  – Managing access control on directories and objects, and monitoring the size and usage of directories in the server's clearinghouse. Creating new directories, possibly with the namespace administrator, when necessary.
  – Creating new objects in directories or overseeing their creation. (Beyond a certain level in the directory hierarchy, you also can delegate the responsibility of maintaining directories and the objects in them.)
  – Adding new administrators to the **cds-admin** security group.

The following chapters in Part 6, “DCE Directory Service” on page 157 provide detailed information about how to perform these tasks:

- Chapter 23, Controlling Access to CDS Names
- Chapter 24, Managing Clerks, Servers, and Clearinghouses
- Chapter 25, Managing CDS Directories
- Chapter 28, Restructuring a Namespace

Cell Directory Service Security and Access Control

The DCE control program (**dcecp**) and the CDS ACL Manager both work to manage authorization in CDS. When a CDS control program request is issued to perform an operation on a CDS object, the CDS ACL Manager checks permissions, based on ACL entries, and grants or denies the request.

To change, add, delete, or view ACL entries in the CDS namespace, use the **dcecp 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 CDS server (**cdsd**) and the CDS Advertiser (**cdsadv**) processes.

The chapter entitled Chapter 23, Controlling Access to CDS Names in Part 6 provides detailed information about handling CDS security and access control, including guidelines for setting up access control in a new namespace.

Distributed Time Service 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:

- Identifying system clock problems
- Adjusting system clocks
- Changing DTS attributes for varying WAN conditions
- Modifying DTS configuration when the network environment changes.

For more detailed information on DTS maintenance tasks, see Part 7, “DCE Distributed Time Service” on page 257.

Managing the Distributed Time Service

You can use the **dcecp** program to create and then enable a DTS entity.

After these tasks are done, you can perform routine management tasks, such as enhancing performance, reconfiguring the network, and changing local time.

Some **dcecp** commands change and improve the performance of your network. The **dts modify** command changes the values of many DTS-related characteristics. The **dts show** command displays the values of DTS-related characteristics at any time. The following are some of the tasks you can accomplish using the DTS commands:

- Displaying or changing the number of servers that must supply time values to the system before DTS can synchronize the system clock
- Displaying or changing the inaccuracy limit that forces the system to synchronize to bring the inaccuracy back to an acceptable level
- Displaying or changing the interval at which you want clock synchronization to occur
- Displaying or changing the reaction to a faulty system clock
- Displaying or changing the settings that indicate how often to query servers.

For more information about these commands and characteristics, see Chapter 31, “Managing the Distributed Time Service” on page 271. It also describes the following tasks:

- 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 global and local DTS servers to CDS, thereby registering them as objects in the namespace.

Modifying the System Time

Sometimes you need to change 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 reference. You can use the **dcecp** program to manage system time.

Note: In the OS/390 DCE implementation, DTS cannot directly change the hardware clock. A software clock exists which acts as an intermediary between DTS and the hardware clock. The software clock reflects the DTS synchronized time which is the sum of the hardware clock and a time differential calculated by DTS. In this book, the term **system time** refers to the software clock on the OS/390 host that is part of the OS/390 DCE implementation.

You use the **clock set** command to accomplish this task by gradually changing the time.

Use the **clock set** command with the **-abruptly** option and the **dts synchronize** command to adjust the system clock and synchronize systems.

Chapter 31, "Managing the Distributed Time Service" on page 271 gives you more information about running and using the **dcecp** DTS commands and about changing the system time.

DCE Security Service Maintenance Tasks

DCE Security Service management tasks include:

- Creating and maintaining accounts using the **dcecp** program

  The **dcecp** program provides commands to create and maintain registry information, including principals, groups, and accounts. For details on how to use the **dcecp** program for the Security Service, see Chapter 35, "Control Programs for Managing the Security Service" on page 327.

  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 registry connect** command to create an account for a foreign cell.

  When you add new user accounts, and one or more of those users is to be cross linked to RACF, remember to run the RACF interoperability utility, **mvsexpt**, so that the new users will have single sign-on capability and interoperability between RACF and OS/390 DCE. For more information, see "Cross Linking Existing DCE Users who are New RACF Users" on page 405.

- Using Access Control Lists (ACLs)

  Use the **dcecp acl** commands to display, add, change, and delete ACL entries for a specific object in the CDS namespace. See Chapter 34, "Using Access Control Lists" on page 307 for detailed information on how to use the **dcecp acl** commands.

- Setting and maintaining registry policies.

  Registry policies include certain password and account information. Use the **dcecp registry** commands to set and maintain registry policies. Details on using these commands are in Chapter 42, "Maintaining Policies and Properties" on page 411.

  Ticket expiration date, password life span, 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.

- 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** commands to manage Audit

Service data. The *OS/390 DCE: Command Reference* provides descriptions of audit-related **dcecp** objects and commands. See Chapter 48, “DCE Audit Service Administrative Tasks” on page 457 for more information about Audit Service administration.

- Adopting registry objects that are “orphaned” because their owner has been deleted.

  Chapter 45, “Adopting Registry Orphans” on page 427 describes how registry orphans can be adopted.

- Notifying a user to update his or her DCE password in the RACF DCE segment

  When a DCE user is cross linked with RACF and is enabled for single sign-on, the user must update his or her password in the RACF DCE segment when the DCE password changes. This is done by using the **storepw** command. (The password is also updated in the DCE registry if the **-r** option is specified.) Notify users that they must do this before invoking a DCE application, and that each user's principal must be in the security manager (such as RACF) before **storepw** can update the user's DCE registry. For more information on the **storepw** command, see the *OS/390 DCE: Command Reference*.

Reconfiguring the Registry

There are two main reconfiguration tasks included in the administration of the DCE Security Service:

- Changing the master registry site if you plan to move the machine that runs the master 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 slave registry server from the network or shut that machine down for an extended period

Improving Registry Performance in Large Cells

The DCE configuration program creates a host profile, called **/.:/hosts/***hostname***/profile**, for each machine. The default entry in this host profile points to the cell profile.

DCE needs a proper search order to find a security replica to bind to. The _EUV_USE_HOST_PROFILE environment variable allows the DCE administrator to tailor the search order on an individual machine basis.

One way to tailor the search order is to change the default entry in the host profile to point to an area profile that the administrator has created and to set _EUV_USE_HOST_PROFILE to 1. The administrator then adds entries for the security replicas in the cell to this area profile. When an application attempts to bind to a security replica, the replicas in the area profile are tried in priority order.

These are the **dcecp** commands to create a group of security replicas:

```
rpcgroup create /.:/group-name
rpcgroup add /.:/group-name -member {
   /.:/subsys/dce/sec/replica-1
   /.:/subsys/dce/sec/replica-2
         .
         .
         .
  /.:/subsys/dce/sec/replica-n }
```

These are the **dcecp** commands needed to create an area profile:

```
rpcprofile create /.:/area-name
rpcprofile add /.:/area-name -priority 1
      -member /.:/group-name -annotation {rs_bind}
      -interface {d46113d0-a848-11cb-b863-08001e046aa5 2.0}
rpcprofile add /.:/area-name -priority 1
      -member /.:/group-name -annotation {secidmap}
      -interface {0d7c1e50-113a-11ca-b71f-08001e01dc6c 1.0}
rpcprofile add /.:/area-name -priority 1
      -member /.:/group-name -annotation {krb5rpc}
      -interface {8f73de50-768c-11ca-bffc-08001e039431 1.0}
rpcprofile add /.:/area-name -priority 1
      -member /.:/group-name -annotation {rpriv}
      -interface {b1e338f8-9533-11c9-a34a-08001e019c1e 1.0}
rpcprofile add /.:/area-name -priority 1
      -member /.:/group-name -annotation {login}
      -interface {1077f9fe-2060-11d0-8b42-08005acd34e8 1.0}
```

Repeat the five **rpcprofile add** commands for each group to be added to the area profile.

If you need to refer to entries in the cell profile, add an entry in the area profile with the desired interface specification as defined in the cell profile. For example, if you configure a global time server, you need an entry in the area profile for the LAN Services interface.

Improving dce_login Performance

If your installation has many DCE clients logging into DCE from OS/390, the performance of these logins can be improved by using Temporary File Systems (TFS). A TFS is an in-memory hierarchical file system that delivers high-speed I/O. During the DCE login process, files are created in the **/opt/dcelocal/var/security/creds directory**, and one may be created in the **/opt/dcelocal/var/security/preauth** directory. If TFS file systems are mounted on these two directories the creation and use of these files can take advantage of TFS high-speed I/O. In order to have TFS file systems mounted each time your installation is initialized, two control files must be updated:

- The BPXPRM*xx* member in SYS1.PARMLIB
- The HFS file **/etc/rc**.

The statements added to the BPXPRM*xx* member cause TFS file systems to be mounted each time the system is initialized. The statements in the initialization shell script /etc/rc modify the owners and permissions of the mounted directories to meet the requirements of DCE. The following statements can be added to the BPXPRM*xx* member to mount two TFS file systems, one at **/opt/dcelocal/var/security/creds**, and the other at **/opt/dcelocal/var/security/preauth**. The statement that mounts the file system with the mountpoints **/opt/dcelocal/var/security/creds** and **/opt/dcelocal/var/security/preauth** must appear in the BPXPRM member prior to these statements.

```
      MOUNT FILESYSTEM('/TFS/SECURITY/CREDS')
              TYPE(TFS)
              MODE(RDWR)
              MOUNTPOINT('/opt/dcelocal/var/security/creds')
              PARM('-s 400')
      MOUNT FILESYSTEM('/TFS/SECURITY/PREAUTH')
              TYPE(TFS)
              MODE(RDWR)
              MOUNTPOINT('/opt/dcelocal/var/security/preauth')
              PARM('-s 400')
```

The following sample statements in **/etc/rc** change the owner and permission bits on the directories **/opt/dcelocal/var/security/cred** and **/opt/dcelocal/var/security/preauth** to those required by DCE.

```
        # Set owners and permissions for DCE login directories.

        chown 0:0  /opt/dcelocal/var/security/preauth
        chmod 0711 /opt/dcelocal/var/security/preauth
        chown 0:0  /opt/dcelocal/var/security/creds
        chmod 1777 /opt/dcelocal/var/security/creds
```

In the OS/390 implementation of DCE, a client process retains its DCE login credentials across system initializations. If TFS file systems are used for the DCE **creds** and **preauth** directories this is no longer true. Due to the volatile nature of a TFS, all data stored in it is lost when the system is initialized. DCE clients have to log in to DCE after a system IPL.

Part 2. General Administration Aspects of OS/390 DCE

Chapter 4. OS/390 DCE Administration Commands 29
Using the OS/2® Administration GUI 29
DCE Daemons in OS/390 DCE 29
Administration and User Commands 29
Entering Arguments to OS/390 DCE Commands 30
Running the Administration and User Commands in Batch 31
Commands that Cannot Fit in One Line . . . 32
Command Input and Output Redirection . . 32

Chapter 5. Starting and Stopping OS/390 DCE 33
DCEKERN Address Space 33
Stopping DCEKERN 34
Who Can Start and Stop OS/390 DCE Daemons? 34
Ways of Starting OS/390 DCE Daemons . . . 34
Using DCECONF to Start OS/390 DCE 35
The MODIFY DCEKERN Operator Command . 35
Using MODIFY DCEKERN to Start OS/390 DCE Daemons 35
Order of Starting OS/390 DCE Daemons . . 35
Using MODIFY DCEKERN to Stop OS/390 DCE Daemons 36
Viewing the Status of DCE Daemons 36
Starting OS/390 DCE During System IPL . . . 37
Daemon Configuration File 37
How DCEKERN Starts the DCE Daemons . . 38
Using the -nodce Option to Start DCEKERN 38
Stopping and Restarting the CDS Advertiser and Clerk Daemons 38
Controlling Where Daemons are Started 39
Using the Control Task to Monitor HFS Utilization by DCE 39

Chapter 6. Structure and Backup of OS/390 DCE Files 41
Global Files 41
Local Files 41
Backup Strategy 42
Utilities for Backing Up Files 42
Mapping of HFS to PDS 42

Chapter 4. OS/390 DCE Administration Commands

This chapter describes OS/390 aspects of running the OS/390 DCE administration commands.

Using the OS/2® Administration GUI

There is a DCE administration graphical user interface (GUI) available in IBM Directory and Security Server for OS/2 Warp Version 4. This is the group of products containing DCE for OS/2; the GUI is available through any of the products. Order numbers are:

- IBM Directory and Security Server for OS/2 Warp Version 4 DES Version, 10H9754
- IBM Directory and Security Server for OS/2 Warp Version 4 non-DES Version, 25H7945
- Distributed Computing Environment Client including Distributed File System Version 4 DES Version, 25H7940
- Distributed Computing Environment Client including Distributed File System Version 4 non-DES Version, 25H7946

If you have this product or are planning to order it, you should know that it can be used to administer OS/390 as both client and server in a DCE cell. The OS/390 Security Server can also be present in the same DCE cell.

DCE Daemons in OS/390 DCE

OS/390 DCE provides the DCE client daemons that allow the OS/390 host to participate in DCE. Following are the DCE daemons available in OS/390 DCE:

- DCE Daemon
- OS/390 Security Server Daemon (available as an optional feature of OS/390)
- CDS Advertiser
- CDS Clerk
- CDS Server Daemon
- DTS Null Time Provider Daemon
- DTS Daemon
- Audit Daemon
- Password Management Daemon
- GDA Daemon

Administration and User Commands

In OS/390 DCE, the administration and user commands can be run from TSO, the OS/390 shell, or submitted as a batch job. For simplicity, most of the examples in this book are in command line or interactive modes only. That is, commands are *entered* from TSO or from the shell. There is a slight difference in some command names used to run these facilities when running from TSO (or batch) and from the shell.

While OS/390 DCE command names can be entered in either upper- or lowercase in TSO (or batch), these commands can only be entered in lowercase in the OS/390 shell. Also, some OS/390 DCE commands in the shell contain underscores. You must enter all OS/390 DCE commands in TSO without underscores.

Table 3 on page 30 lists these facilities and the corresponding command names for running them in TSO, batch, and the shell.

In batch, these names correspond to the PROC names that are shipped with the OS/390 DCE product for these facilities.

Table 3. OS/390 Administration Commands

Facility	**TSO and Batch**	**OS/390 Shell**
ACL Editor	ACLEDIT	acl_edit
CDS Control Program	CDSCP	cdscp
DCE Control Program	DCECP	dcecp
DTS Control Program	DTSCP	dtscp
Login Activity	loginact	login_activity
Registry Editor	RGYEDIT	rgy_edit
RPC Control Program	RPCCP	rpccp

Table 4 lists the user commands and the corresponding names for running them in TSO, batch and the shell.

Table 4. OS/390 DCE User Commands

Command Description	**TSO and Batch**	**OS/390 Shell**
DCE Login	DCELOGIN	dce_login
Destroy Login Context	KDESTROY	kdestroy
List Kerberos Tickets	KLIST	klist
Refresh Credentials Cache	KINIT	kinit

User commands are described in *OS/390 DCE: User's Guide*.

Entering Arguments to OS/390 DCE Commands

Do not enter administrative or user commands that use all uppercase or mixed-cased arguments from the ISPF command line. Aside from the parameters that are required by these commands, the term **argument** can also be subcommands of the DCE, RPC, CDS, and DTS control programs, and the Registry Editor. This can also be object pathnames that are required by the ACL Editor. When entered from the ISPF command line, arguments that are all uppercase or are of mixed case are converted to all lowercase characters.

This will create a problem for arguments (say, CDS directory names) that have mixed or all uppercase characters. For example, entering the following command:

```
tso cdscp create dir /.:/TestDir
```

from the ISPF command line will actually create the directory /.:/testdir.

If the administrative command has all uppercase or mixed-cased arguments, enter it from the OS/390 shell or from native TSO only.

In the case of DCECP, RPCCP, CDSCP, DTSCP, DCELOGIN, and the Registry Editor, you can also run the command without any arguments from the ISPF command line to start an interactive session. Once in the interactive session, you can enter all uppercase, all lowercase, or mixed-case arguments to these commands.

Running the Administration and User Commands in Batch

In batch, a SYSIN DD statement can point to a file that contains the control program subcommands and the required arguments.

The names of the PROCS to run these commands that are shipped with the OS/390 DCE product are listed in the second column of Table 3 and Table 4.

For example, to run the DCECP server command:

server show /.:/hosts/cellname/config/srvrconf/testsrvr

the following JCL passes the control program commands and arguments in the parameter (PARM) field:

```
//* JCL TO EXECUTE THE DCECP SERVER SHOW COMMAND
//JOB1  JOB...
//GO       EXEC   PROC=DCECP,
// PARM='/-c server show /.:/hosts/cellname/config/srvrconf/testsrvr'
```

Following is an example of a control program command that is run using an inline data set. DCECP processes the command as though it was run interactively.

```
//* JCL TO EXECUTE THE DCECP SERVER SHOW COMMAND
//JOB1  JOB...
//GO       EXEC   PROC=DCECP
  .
  .
//SYSIN    DD *
server show /.:/hosts/cellname/config/srvrconf/testsrvr
/*
```

In the following example, the JCL refers to a data set name that contains the DCECP command and arguments. DCECP processes the command as though it was run interactively.

```
//* JCL TO EXECUTE THE DCECP SERVER SHOW COMMAND
//*
//JOB1  JOB...
//GO       EXEC   PROC=DCECP
.
.
//SYSIN    DD     DSN=DCECP.INPUT,DISP=SHR
```

In this example, the data set DCECP.INPUT will contain the following statement:

server show /.:/hosts/cellname/config/srvrconf/testsrvr

Notes:

1. Control program subcommands and arguments must be in the local code page when they are passed by any of the following:

   - A SYSIN DD statement pointing to a file
   - A SYSIN DD statement pointing to an inline data set
   - The parameter (PARM) field

   This is because the control program command processes the subcommands as though they were run interactively.

2. When you are creating JCL, be sure that there is no data in columns 72-80.

The _EUV_ECHO_STDIN Environment Variable: You can set the **_EUV_ECHO_STDIN** environment variable to **1** to display the invocation of the administrative or user command in the standard output file.

This is especially useful when running the commands in batch, where the output file will give you an indication on which commands failed, if any should occur.

Setting environment variables is discussed in Appendix A, "Environment Variables in OS/390 DCE" on page 467.

Commands that Cannot Fit in One Line

When entering administrative or user commands interactively, the command may exceed one line (255 characters). If the command exceeds one line, use the back slash character (\) at the end of the present line to continue to the next line.

Command Input and Output Redirection

Like any OS/390 DCE program, the DCE administration and user commands can get their input from, or redirect its output to, a file (both HFS and PDS). For example, the Registry Editor can get its input from a file that contains the Registry Editor subcommands as follows:

```
rgy_edit < inputfile
```

Here, **inputfile** may contain the following subcommands that create DCE principals:

```
domain principal
add princ1
add princ2
add princ3
add princ4
```

File redirection is discussed in more detail in *OS/390 UNIX System Services User's Guide*.

The DCE command program syntax allows an input file to be directly specified without redirection:

```
dcecp inputfile
```

In this case *inputfile* is the name of a script file containing **dcecp** and Tcl commands.

Chapter 5. Starting and Stopping OS/390 DCE

This chapter briefly describes the **DCEKERN** address space, then discusses the various ways of starting and stopping OS/390 DCE daemons.

DCEKERN Address Space

In OS/390 DCE, the DCE daemons are controlled by the Control Task running in the DCE Kernel address space (also called the DCEKERN address space). All requests to start or stop the DCE daemons, either collectively or individually, are made through the Control Task. The **dced**, **cdsadv**, **cdsclerk**, and **dtsd** daemons run as child processes of the Control Task, within the DCEKERN address space. Each of the rest of the daemons (**secd**, **cdsd**, **dtstp**, **auditd**, **pwdmgmt**, and **gdad**) can run either as a child process of the Control Task within the DCEKERN address space, or as a process in its own address space.

By default, **secd** and **cdsd** run in their own address spaces, while the others run as child processes of the Control Task in the DCEKERN address space. The default structure of the daemons is shown in Figure 7. The dashed lines indicate daemons that run in their own address spaces but are controlled by the Control Task in the DCEKERN address space.

To override the default, specify the environment variable, _EUV_DAEMONS_IN_AS, in the **envar** file for DCEKERN, **/opt/dcelocal/home/dcekern/envar**. For details on controlling where daemons start, see “Controlling Where Daemons are Started” on page 39.

After the DCEKERN address space is configured, all the daemons that have been configured are started automatically when the DCEKERN address space is started. The Control Task and all daemons run as **root**.

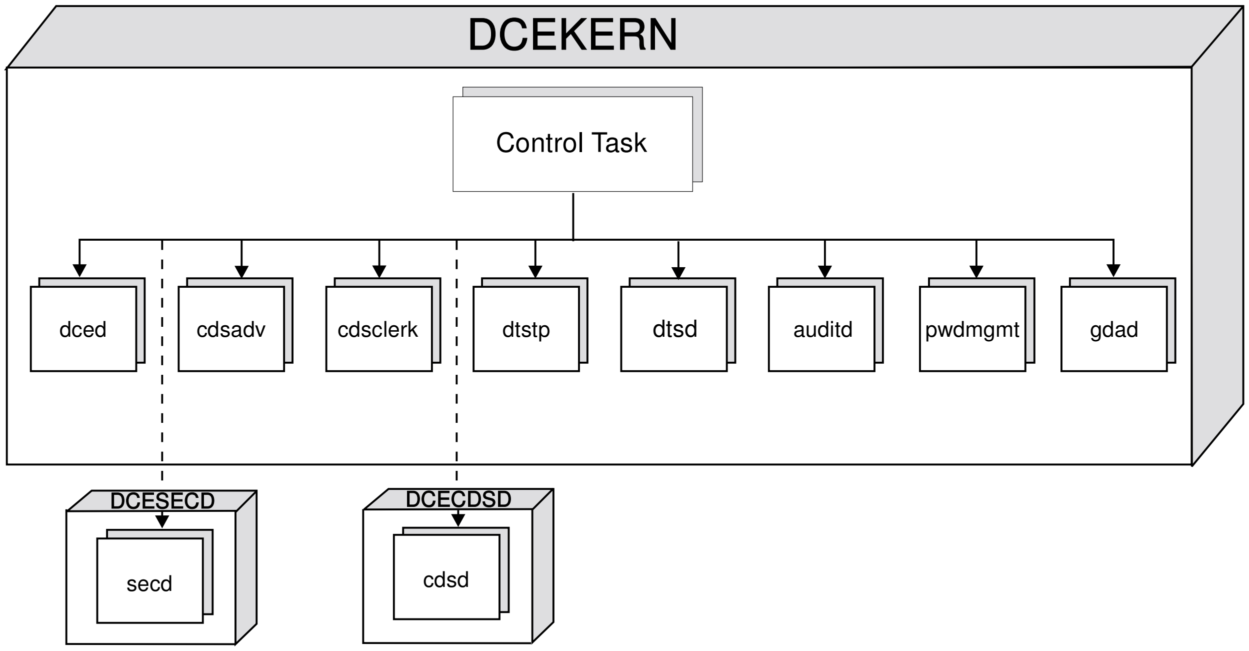


Figure 7. The DCEKERN Address Space

All requests to DCEKERN are directed to the Control Task that performs the requested action. The DCE daemons can be started and stopped using an operator command. In starting and stopping the daemons, the Control Task uses a **Daemon Configuration File** that contains information on the daemons that were previously configured on the host. The Daemon Configuration File contains runtime options, startup parameters, and restart information for each daemon. For details on the Daemon Configuration File, see “Daemon Configuration File” on page 37.

Besides starting and stopping the DCE daemons, the Control Task can also detect daemons that have prematurely stopped and tries to restart them automatically. The algorithm used by DCEKERN in starting and restarting the OS/390 DCE daemons is summarized in “How DCEKERN Starts the DCE Daemons” on page 38.

Note: If there is an existing endpoint map file on the host system, you must delete it before starting the DCEKERN address space. You can remove the endpoint map file, **/opt/dcelocal/var/adm/rpc/rpcdepa.dat**, during the startup of OS/390 by including a **rm /opt/dcelocal/var/adm/rpc/rpcdepa.dat** command in the **/local/bin/rc.local** file. This file is called during OS/390 initialization through **/etc/rc**.

Stopping DCEKERN

To stop the DCEKERN address space, use the **stop** operator command to ensure the normal shutdown of the address space.

In the event that you must use a **cancel** operator command to stop the DCEKERN address space, an OS/390 system ABEND S069 will occur in any DCE daemon processes that are running in their own address spaces. This condition is expected and can be ignored by the operator.

Who Can Start and Stop OS/390 DCE Daemons?

There are two types of users who can start or stop the DCE daemons in OS/390 DCE:

- A user with OS/390 operator privileges.
- A user who has update privilege to the **DCEKERN.START.REQUEST** RACF® facility. This facility is created during the installation of OS/390 DCE. For more information on this RACF facility, refer to *OS/390 Program Directory*.

Ways of Starting OS/390 DCE Daemons

OS/390 DCE daemons are started in any of the following ways:

- Using the configuration program, DCECONF
- Using the MODIFY DCEKERN operator command
- During the system IPL.

The OS/390 DCE daemons are configured and started during the configuration of the host system, using the DCECONF program.

Ideally, the daemons run continuously in the background and do not need to be started or stopped again. However, the DCE daemons may have to be started manually in certain situations, for example, if a daemon ends abnormally. You can use the MODIFY operator command to manually start or stop the DCE daemons.

You can also have the DCE daemons started automatically during the system IPL.

Each of these alternatives is discussed in the following sections.

Using DCECONF to Start OS/390 DCE

After installing OS/390 DCE, you need to run DCECONF, a menu-driven program, to configure and start the DCE daemons. DCECONF performs the following:

- Creates all necessary configuration files, security principals, and namespace entries
- Deletes old configuration files
- Starts the DCE daemons.

DCECONF is described in detail in *OS/390 DCE: Configuring and Getting Started*.

The MODIFY DCEKERN Operator Command

The DCE daemons can be started or stopped using the **MODIFY DCEKERN** operator command. Using MODIFY DCEKERN, you can also view the status of the DCE daemons. For details on the MODIFY DCEKERN command options, refer to *OS/390 DCE: Command Reference*.

Using MODIFY DCEKERN to Start OS/390 DCE Daemons

With the MODIFY DCEKERN operator command, you have the option of starting an individual daemon or starting all the daemons using a single command.

For example, to start the DCE host daemon, enter the following:

```
MODIFY DCEKERN, start dced
```

To start all the daemons, enter the following:

```
MODIFY DCEKERN, start all
```

Note: Do not use the MODIFY command to start the OS/390 DCE daemons while the DCEKERN address space is still initializing. During initialization, DCEKERN will attempt to start all the OS/390 DCE daemons that have been configured on the OS/390 host. If you enter the MODIFY command while DCEKERN is initializing, the OS/390 DCE daemons may be started out of order or stopped erroneously. This may lead to unexpected errors during initialization and cause DCEKERN to end abnormally.

You must wait until DCEKERN has issued a log message indicating that DCEKERN initialization has completed before using the MODIFY commands. The log messages are sent to the operator's console log.

Order of Starting OS/390 DCE Daemons

When DCE daemons are started manually, the successful startup of some daemons depends on the availability of the services provided by other daemons. This implies that the DCE daemons must be started in a particular order.

Following is the sequence by which OS/390 DCE daemons should be started.

Note: This is applicable *only* if you need to start any of the OS/390 DCE daemons individually. If the OS/390 DCE daemons are started collectively (for example, using the **start all** option of the MODIFY DCEKERN command), DCEKERN ensures that the correct starting sequence is followed.

1. dced
2. secd
3. cdsadv
4. cdsclerk
5. cdsd
6. dtstp
7. dtsd
8. auditd
9. pwdmgmt
10. gdad

For example, to successfully start the cdsadv daemon, the dced daemon must already be up and running. One or more messages are issued for each daemon when it starts. Look for these messages in the operator's console log.

Note: Make sure that the passwords of the OS/390 DCE daemons are valid before starting them up. If the passwords have expired, the daemons cannot be started. This is discussed in “If OS/390 DCE Daemons' Passwords Expire” on page 348.

Using MODIFY DCEKERN to Stop OS/390 DCE Daemons

You can use the **MODIFY DCEKERN** system command to stop a DCE daemon or all daemons that are configured on the host.

For example, to stop the DCE host daemon, enter:

```
MODIFY DCEKERN, stop dced
```

To stop all daemons on the host, enter:

```
MODIFY DCEKERN, stop all
```

Note: IBM recommends that you stop all DCE servers before stopping DCEKERN to avoid the possibility of 0D6 abends and the resulting dumps.

Viewing the Status of DCE Daemons

You can query the status of the DCE daemons using the **query** option of the MODIFY system command. You do not need the special privileges of a DCE administrator or an operator to use the **query** option.

For example, to query the status of the DCED host daemon, enter the following:

```
MODIFY DCEKERN, query dced
```

A message about the status of the daemon will be written on the system log. This message will also contain the **process ID** of the daemon.

The status of the daemon can be any of the following:

READY	Indicates that the daemon is running, has been initialized, and is ready to receive and process incoming requests.
INITIALIZING	Indicates that the daemon has been started, but is not yet ready to receive and process incoming requests.
STOPPING	Indicates that a request to stop the daemon has been received and that the daemon is in the process of stopping.
DOWN	Indicates that the daemon is not active.

UNKNOWN Indicates that the status of the daemon cannot be determined. This can occur if the daemon was started, but no response was received by the system indicating a change in its status.

Note: You can enter a command to stop a daemon *only* if it is in the READY state or in the UNKNOWN state.

Starting OS/390 DCE During System IPL

Because the OS/390 DCE daemons are contained in the DCEKERN address space, these daemons are started during the initialization of DCEKERN. This lets you configure the host to automatically start the DCEKERN address space during the system IPL.

The Control Task of DCEKERN uses the Daemon Configuration File to determine which daemons can be started, and the parameters to pass to the daemon load module when starting the daemon.

Daemon Configuration File

The Daemon Configuration File is used by the Control Task to obtain necessary information when starting the DCE daemons. Information on this file is initially set by the initial configuration program, DCECONF. The Daemon Configuration File contains the following information:

- The DCE daemons that are configured on the host. Only the daemons that are configured on the host using the DCECONF program can be started.
- Parameters that are passed to the load module when a daemon is started, called the **argument list** (including LE/370 runtime options).
- The **Minimum Restart Interval**. The Control Task attempts to restart a daemon that ends abnormally only if the daemon was running for at least this time interval. If a daemon ends during this time interval, it will not be restarted.
- The **Time-out Period**, which is the maximum time interval that the Control Task waits for the daemon to complete its initialization after it has been started. When this time interval elapses, and the Control Task has not received confirmation from the daemon that initialization has completed, the status of the daemon is set to **UNKNOWN**.

The pathname to the Daemon Configuration file is **/opt/dcelocal/etc/euvpdcf**. Figure 8 shows the typical contents of the Daemon Configuration file.

```
DCED     CONFIGURED=N LMD=EUVDCED  ARG="ENVAR('_EUV_HOME=/opt/dcelocal/home/dced')/                >DD:DCEDOUT" RESTART=300 TIMEOUT=300
SECD     CONFIGURED=N LMD=EUVSSECD ARG="ENVAR('_EUV_HOME=/opt/dcelocal/home/secd')/-dcekern        >DD:SECDOUT" RESTART=300 TIMEOUT=300
CDSD     CONFIGURED=N LMD=EUVCCDSD ARG="ENVAR('_EUV_HOME=/opt/dcelocal/home/cdsd')/                >DD:CDSDOUT" RESTART=300 TIMEOUT=300
CDSADV   CONFIGURED=N LMD=EUVCADV  ARG="ENVAR('_EUV_HOME=/opt/dcelocal/home/cdsadv')/              >DD:ADVOUT"  RESTART=300 TIMEOUT=300
CDSCLERK CONFIGURED=N LMD=EUVCCLRK ARG="ENVAR('_EUV_HOME=/opt/dcelocal/home/cdsclerk')/            >DD:CLRKOUT" RESTART=300 TIMEOUT=300
DTSD     CONFIGURED=N LMD=EUVTDTSD ARG="ENVAR('_EUV_HOME=/opt/dcelocal/home/dtsd')/-r              >DD:DTSDOUT" RESTART=300 TIMEOUT=300
DTSTP    CONFIGURED=N LMD=EUVTNP   ARG="ENVAR('_EUV_HOME=/opt/dcelocal/home/dts_null_provider')/ >DD:TNPOUT"  RESTART=300 TIMEOUT=300
AUDITD   CONFIGURED=N LMD=EUVSAUDD ARG="ENVAR('_EUV_HOME=/opt/dcelocal/home/auditd')/              >DD:AUDOUT"  RESTART=300 TIMEOUT=300
PWDMGMT  CONFIGURED=N LMD=EUVSPWD  ARG="ENVAR('_EUV_HOME=/opt/dcelocal/home/pwdmgmt')/-dcekern     >DD:PWDOUT"  RESTART=300 TIMEOUT=300
GDAD     CONFIGURED=N LMD=EUVCGDAD ARG="ENVAR('_EUV_HOME=/opt/dcelocal/home/gdad')/                >DD:GDADOUT" RESTART=300 TIMEOUT=300
```

Figure 8. Daemon Configuration File

In the **ARG** (argument list) field of this example, **ENVAR** indicates the environment variables to be used, in this case, the **_EUV_HOME** variable which points to the daemon's home directory. Anything after the ")/" characters are program parameters. The ">" character is the redirection character which indicates that the output will be redirected to the symbolic name of the DD statement that follows.

Important

Under usual circumstances, you do **not** need to edit the Daemon Configuration File. This file is managed by the DCE configuration program.

How DCEKERN Starts the DCE Daemons

When a request arrives to start DCE daemons, DCEKERN looks at the Daemon Configuration File to see if the particular daemon or daemons are configured on the host. If the daemon is configured and is not running, DCEKERN starts it and waits for the daemon to initialize successfully.

In case of the abnormal ending of any DCE daemon, DCEKERN, through the Control Task, tries to restart the daemon. The Control Task will attempt to restart the daemon only if the daemon was running for at least the duration of the Minimum Restart Interval, as specified in the Daemon Configuration File. If the daemon ended within this time interval, it will not be restarted.

There is a special case when the CDS Clerk ends abnormally. DCEKERN stops the CDS Advertiser, then restarts it before starting the CDS Clerk.

Note: When DCEKERN restarts an abnormally terminated daemon, it does not correct the problem that caused the daemon to end unexpectedly. Thus, depending on the cause of the abnormal ending, the daemon may be unsuccessful again after restarting because of the same error condition.

Using the -nodce Option to Start DCEKERN

You can start the DCEKERN address space without starting the configured DCE daemons by using the **-nodce** option of the **start** (**/s**) operator command, for example:

```
/s dcekern,parms='-nodce'
```

This option is especially useful when the configuration program (DCECONF) failed in midstream, and you have to restart the DCEKERN address space prior to reconfiguration of the host.

Restarting DCEKERN When CDS Cache Files are Deleted: Another instance when you have to start the DCEKERN using the -nodce option is if the CDS cache files are deleted and you want to restart the DTS daemon. In this case, start DCEKERN (using the -nodce option), then manually start each DCE daemon. Before starting the DTS daemon, you must use the **dcecp cdscache create** command to add the knowledge of the server to the cache. For more information on this command, see Chapter 7, “DCE Control Program Introduction” on page 45 and the *OS/390 DCE: Command Reference*.

Stopping and Restarting the CDS Advertiser and Clerk Daemons

There are special considerations when stopping and restarting the CDS Advertiser and Clerk daemons. These are discussed in detail in “Stopping the CDS Advertiser and CDS Clerk” on page 209.

Controlling Where Daemons are Started

You can use the environment variable, _EUV_DAEMONS_IN_AS, to control whether a daemon is started in its own address space or is started within the DCEKERN address space. Running a daemon in its own address space may reduce contention for resources in the DCEKERN address space.

When added to the **/opt/dcelocal/home/dcekern/envar** file, the variable specifies the list of daemons to be started in their own address spaces. The list can contain none or any of the following daemons: **secd**, **cdsd**, **dtstp**, **auditd**, **pwdmgmt**, or **gdad**. For example,

```
_EUV_DAEMONS_IN_AS=auditd cdsd secd
```

causes only these three daemons to start in their own address spaces.

If the _EUV_DAEMONS_IN_AS variable is not specified, the default is to run only **secd** and **cdsd** in their own address spaces. Therefore, to override the default and run *all* daemons within the DCEKERN address space, the environment variable should be

```
_EUV_DAEMONS_IN_AS=N
```

Note that some of the daemons (**dced**, **cdsadv**, **cdsclerk**, and **dtsd**) only run within the DCEKERN address space. They are ignored if specified in the _EUV_DAEMONS_IN_AS variable.

If you want to change where a daemon is started, modify the value of the _EUV_DAEMONS_IN_AS variable, stop DCEKERN if it is running, and then restart DCEKERN.

Note: In OS/390 DCE, all environment variable (**envar**) files must be in code page IBM-1047, so remember to use this code page when updating the **envar** files.

Using the Control Task to Monitor HFS Utilization by DCE

The Control Task can assist administrators in monitoring the use of Hierarchical File System (HFS) space by DCE. It can, at specified intervals, check mounted file systems used by DCE and display a message if any file system exceeds a specified utilization percentage (threshold).

The monitoring is enabled by setting the environment variable, _EUV_HFS_MON, in the **/opt/dcelocal/home/dcekern/envar** file to the desired monitoring time interval. The interval may be specified in hours (H) or minutes (M), up to **596 523** hours or **35 791 394** minutes. If _EUV_HFS_MON is not defined, no monitoring will be done.

An example of this environment variable is:

```
_EUV_HFS_MON=1H
```

This tells the Control Task to perform the test every hour.

The HFS mount points to be monitored by the Control Task are specified in the file **/opt/dcelocal/home/dcekern/euvhfsmon**. This file contains one or more records in the format:

```
utilization_threshold_percentage:hfs_file_system_path_name
```

where *utilization_threshold_percentage* is an integer representing the level of utilization of the associated *hfs_file_system_path_name* at which the Control Task displays a message indicating that this threshold has been reached or exceeded.

For example, if the file contains the single record:

```
90:/opt/dcelocal
```

the control task displays the following message when **/opt/dcelocal** is filled to 95% of its capacity at the time a monitoring interval occurs:

```
EUVP00169I  Current hfs utilization percentage of 95 exceeds threshold
percentage of 90 for path /opt/dcelocal.
```

Chapter 6. Structure and Backup of OS/390 DCE Files

This chapter describes the structure of the OS/390 DCE files and recommends a back up strategy based on the nature of the files. It assumes that you are familiar with UNIX System Services file systems.

All of the DCE HFS files are either in the **/usr/lpp/dce**, **/opt/dcelocal**, or **/krb5** directories. The **/usr/lpp/dce** directory contains static read-only files that can be shared by multiple systems. The **/opt/dcelocal** directory contains read-write files that are unique to each system. The **/krb5** directory contains **Kerberos** configuration files.

Global Files

The **/usr/lpp/dce** file system can be mounted by multiple systems running the same level of OS/390 if the HFS data sets are mounted in read-only mode. This allows maintenance to be applied only once to all the systems.

Following are the subdirectories within **/usr/lpp/dce**:

- **/usr/lpp/dce/bin**

  Contains the user and administrative utilities: DCE control program, Registry Editor, ACL Editor, RPC control program, CDS control program, DTS control program, DCE-RACF cross-linking utilities (**mvsimpt** and **mvsexpt**), dce_login, kinit, kdestroy, and klist.

- **/usr/lpp/dce/lib**

  Contains library stubs used by application developers.

- **/usr/lpp/dce/lib/nls**

  Contains National Language Support files (for example, message catalog files).

- **/usr/lpp/dce/share/include**

  Contains header files used by application developers.

- **/usr/lpp/dce/examples**

  Contains examples of DCE applications.

Local Files

The **/opt/dcelocal** file system is connected by a symbolic link to **/etc/dce**. This file system cannot be shared across systems. Each system must have its own copy. All files unique to a system are stored in HFS files. None are stored in OS/390 data sets. Following are the subdirectories within **/opt/dcelocal**:

- **/opt/dcelocal/etc**

  Contains the local DCE configuration files maintained by the OS/390 host system like the CDS attributes file and the daemon configuration file. Maintenance that is applied to any file in this file system must be manually propagated to all other systems.

- **/opt/dcelocal/var**

  Contains the data files that are maintained by the OS/390 DCE daemons.

- **/opt/dcelocal/svc**

  Contains files related to the OS/390 DCE serviceability subsystem.

- **/opt/dcelocal/home**

Contains the home directories of the DCE daemons.

- **/opt/dcelocal/dcecp**

  Contains the script files for **dcecp** initialization and for some **dcecp** commands.

- **/opt/dcelocal/tcl**

  Contains the script files for **tcl** initialization.

Backup Strategy

The local files are dynamic files and are candidates for regularly-scheduled backup. The global files are static files and need only be backed up as part of the full set of target libraries and their associated SMP/E zones.

Utilities for Backing Up Files

The Data Facility Data Set Services (DFSMSdss™) utility of Data Facility System Managed Storage (DFSMS/MVS®) can be used to backup or restore these file systems.

Note that, with DFSMSdss, you can only back up whole file systems. There are pax and tar utilities that allow you to back up specific files within a file system. The pax and tar utilities should be used only for user directory backup.

Mapping of HFS to PDS

When using DFSMSdss Logical Dump to back up files in HFS file systems, you must use the name of the HFS data set that contains the directories you want to back up or restore. You can also obtain the HFS-to-PDS/EX file system mapping in the **BPXPRMXX** member of PARMLIB.

For more information on using DFSMSdss, refer to the *Data Facility Data Set Services User's Guide*. For more information on using the **pax** and **tar** commands, refer to the *OS/390 UNIX System Services User's Guide*.

Part 3. The DCE Control Program

Chapter 7. DCE Control Program Introduction 45
Flexible, Portable, and Extensible Administration 45
DCE Administration Objects 46
Using the DCE Control Program 47
Starting and Stopping the dcecp Program . 47
Invoking dcecp Operations 47
Disabling the Alternate operation object Syntax 49
Using Global Error Information Variables . . 49
Doing More with dcecp 50
When To Use an Interactive Command or Script 52
Editing Command Lines 52
Editing Command Lines with the history Command 53
Using the dcecp Help Facilities 54
Customizing dcecp Sessions 56
Adding Scripts to dcecp Sessions 57
Adding New Objects to the DCE Control Program 59
Convenience Variables Mean Fewer Keystrokes 59
Current Principal (User) Name (_u) 60
Current Cell Name (_c) 60
Current Host Name (_h) 60
Most Recent Operation Argument Name (_n) 60
Parent of the _n (_p) 61
Last dcecp Object Name (_o) 61
Last Operation's Return Value (_r) 62
DCE Servers to Use (_s(xxx)) 63
Last Security Server Used (_b(sec)) 64
The proc Command Lets You Create New Commands 64
Running Operating System Commands from a Script 66

Chapter 8. Writing Scripts and dcecp Objects 69
Informal Administration Scripts 69
Formal Task Objects 71
A Model for Task Objects 71
Using the parseargs Procedure 78
Invoking Task Objects 80

Chapter 7. DCE Control Program Introduction

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 run.

The DCE Control Program (**dcecp**) available with OS/390 DCE Release 1 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.

Flexible, Portable, and Extensible Administration

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. A Tcl command interpreter is provided as part of the DCE Control Program.

Note: The DCE implementation of the Tcl command interpreter is called the DCE control program language. From now on, this document will refer only 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 run 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 run from any platform based on the OSF DCE Version 1.1 allowing administrators to manage just about all DCE operations from any DCE Version 1.1 system in the cell. Non-UNIX 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 needn't 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.

  Note: In OS/390 UNIX System Services DCE, all **dcecp** script files must be in code page IBM-1047. If you port a script file from another platform, be sure that it is converted to code page IBM-1047.

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 operations not implemented are those that are performed only once or relatively infrequently. OSF plans to implement all DCE operations in **dcecp** eventually. For now, you have to use the individual component control programs to perform operations not implemented in **dcecp**.

The chapters in Part 3, “ The DCE Control Program” discuss how you can use 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, see *Tcl and the Tk Toolkit*, John K. Osterhout, (c)1994, Addison-Wesley Publishing Company.

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.

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
1996-09-23-10:46:42.016-04:00I-----
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 *OS/390 DCE: Command Reference*.

Using the DCE Control Program

This section provides a quick look at how to start and stop the DCE control program and how to perform operations. Additional information about these topics is contained in the **dcecp** section of the *OS/390 DCE: Command Reference*.

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 run them directly at the operating system prompt.

If you will be doing several operations, you can start 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.

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

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

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
$
```

Invoking dcecp Operations

If you are performing a single **dcecp** operation, you can run 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
IBM 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/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}
$
```

You can also enter some limited multiple operations using the semicolon (;) as a command separator and enclosing the operations in double quotation marks. 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 display the results of the other commands, use the Tcl **puts** command:

```
$ dcecp -c "puts [group list staff]; puts [group list managers]; group list users"
```

To run a **dcecp** script, omit the **-c** argument but include the name of the script. The following example runs a script named **list_hosts** which lists the names of all hosts in the cell in alphabetic order.

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

When you want to run complex or multiple operations, you might want to do it from within the **dcecp** program. The program provides a convenient history facility that is useful for recalling and reusing previous operations.

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 abbreviations 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.

Disabling the Alternate operation object Syntax

dcecp is configured to accept commands in the order *operation object*, in addition to the standard *object operation* syntax. If you do not want the alternate syntax to be used, edit **/opt/dcelocal/dcecp/tclIndex** and remove all records that contain **source $dir/verb-object.dcp**.

Individual users can still use the alternate syntax by adding the following line to their **.dcecprc** script file in their HOME directory:

```
source /opt/dcelocal/dcecp/verb-object.dcp
```

If you do not want the alternate syntax used at all, edit **/opt/dcelocal/dcecp/tclIndex** and remove all records that contain **source $dir/verb-object.dcp** and then change the permissions on **/opt/dcelocal/dcecp/verb-object.dcp** so it cannot be read, or erase the file.

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
Wrong number of args: must 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.

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 while loop, located at line 60 in the **parseargs** procedure, called from the **_dcp_create_user** procedure of a **user** operation.

```
dcecp> set errorInfo
EUVA04118E Unrecognized 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
    (procedure "user" line 35)
dcecp>
```

The **dcecp** program provides two global variables that store additional error information returned from commands. The **errorInfo** variable contains the **stack_trace** of the error messages, providing the precise location of the error and a trace of what processing was taking place. The **errorCode** variable, if set, usually contains error information from a system function that was called as part of a command (such as **open**).

As with any **dcecp** variable, you can enter **set errorCode** and **set errorInfo** to display their current values. In addition, there are two ways to automatically display the values:

- Set the DCECP_ERROR environment variable to ON. The values of **errorInfo** and **errorCode** (if set) are displayed with the regular output whenever an error is found. This includes errors found by **dcecp**, when started in both command line and interactive mode, and when sourcing script files. DCECP_ERROR can be set before starting **dcecp** or it can be set or unset during **dcecp** processing by changing the value of the **env(DCECP_ERROR)** variable.
- Set the **dcecp_verbose_errors** variable to **1**. The value of **errorInfo** is displayed instead of the regular error output whenever an error is found. **dcecp_verbose_errors** cannot be set before **dcecp** is started. If you want it to be active when starting **dcecp** in command line mode, add it to the source file or command that is being processed.

Doing More with dcecp

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 may 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.

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 **switch** command for testing values against various patterns, and a **proc** command for writing your own customized commands.

The **dcecp** program also includes other syntactic elements such as quotation marks (“”), braces ({}), brackets ([]), and the back slash (\) 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 may find yourself using some constructs and elements (particularly quotation marks, braces, brackets, and back slashes) in interactive operations as well. You 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.

Now let's look at a couple of 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, the account expiration date for the principal (N_Long) named in the command line is being changed. While it is relatively simple to run 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 run 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 runs 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.

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. Usually command arguments are separated by spaces. To prevent **dcecp** from incorrectly interpreting the spaces between list elements as argument separators, braces are used to enclose the list and disable special interpretation of the spaces. Thus all of the list elements appear as one argument. Similarly, braces are used to enclose the individual elements in the script body.

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 back slash (\) line wrap character. The **dcecp** program runs the command when you press <Enter> after the closing brace at the end of line 3.

Now, let's 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'll 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, notice that the **group list temps** operation is in square brackets ([]). Called **command substitution**, this technique 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
```

This chapter has provided a high-level look at some practical uses of **dcecp**. The next chapter looks more closely at some of the **dcecp** operations you are likely to use for DCE administration. Remember that **dcecp** is based on Tcl and Tcl has other commands and command variations not discussed here. So be sure you have access to the standard Tcl publications for detailed information on all of the commands.

When To Use an Interactive Command or Script

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. The following examples typify interactive operations:

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

dcecp> account show w_shakespeare

dcecp> server start /.:/hosts/curley/config/srvrconf/BBSserver
```

Because the next example is a little more complicated, you might choose to run this as a script, at least the first time.

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

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 run scripts, refer to Chapter 8, “ Writing Scripts and dcecp Objects” on page 69.

Editing Command Lines

We've seen some basic ways to enter interactive **dcecp** commands. But let's say that now you want to edit the command you are entering or that you want to recall and modify a command you entered previously. The **dcecp** program offers several ways to recall and modify commands. You can recall and modify a command using the standard OS/390 command retrieval method. Or, use the Tcl **history** command to recall, edit, and reissue a previously used command.

Editing Command Lines with the history Command

Sometimes when you are 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 Tcl **history** command to recall, edit, and reissue a previously used command. The history facility saves only interactive commands. Commands issued from scripts are not saved and cannot 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) on the history list 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
     5  history
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 (1 is the first event) in the list. Negative numbers specify events relative to the most recent command (-1 is the last event). You can also specify an event by typing characters that match all or part of a previous event.

The history facility lets you reuse previous events in many ways. The following discussion covers just a few of the history commands you can use.

- Run 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
            5 history
  dcecp> history redo directory
   .
   . [output of directory show /.:/printers omitted]
   .
  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
 .
 . [output of directory show /.:/printers omitted]
 .
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-run a previous command using the **history substitute** command. The most common use of this command might be to correct typing mistakes. Its syntax is:

  history substitute *old new event_number*

  If you omit the *event_number*, you redo the most recent command. This replaces the *old* part of the recalled command with *new* information and then runs it:

```
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> his sub printer3 printer4 -2
 .
 . [output of object create /.:/printers/ascii_printer4 omitted]
 .
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
 .
 . [output of object create /.:/printers/ascii_printer5 omitted]
 .
dcecp>
```

Using the dcecp Help Facilities

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 as shown in the following example:

```
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


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
dcecp>
```

- 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. The following example shows how to list the operations available for the **principal** object.

  ```
  dcecp> principal operations
  catalog create delete modify rename show help operations
  dcecp>
  ```

  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. The following example returns a one-line description of each operation supported by the **principal** object.

  ```
  dcecp> principal help
  catalog             Returns all the names of principals in the registry.
  create              Creates a DCE principal
  delete              Deletes a principal from the registry.
  modify              Changes the information about a principal.
  rename              Renames the specified principal.
  show                Returns the attributes of a principal.
  help                Prints a summary of command-line options.
  operations          Returns the valid operations for command.
  dcecp>
  ```

- Get information about available command options by adding an *operation* argument to the *object* **help** command. 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 new principal.
  -fullname           Full name of the new principal.
  -quota              Quota of the new principal.
  -uid                User Identifier of the new principal.
  -uuid               Orphaned UUID to be adopted by the specified principal.
  dcecp>
  ```

- Get help about an object itself by using an *object* **help -verbose** command. The following example returns a description of the **principal** object along with information about how to use the object.

```
dcecp> principal help -verbose
This object allows remote manipulation of principal information stored
in the DCE registry.  The argument is a list of either relative or
fully qualified principal names. Specify fixed attributes using
attribute options or attribute lists.  Specify any extended attributes
using attribute lists. Principal operations connect to a registry that
can service the request.  Specify a particular registry by setting the
_s(sec) convenience variable to be a cell-relative or global replica
name, or the binding of the host where the replica exists.  The
completed operation sets _b(sec) to the name of the registry
contacted.  Access to most operations relies solely on the ACL of the
principal.  Create requires i permission to the principal directory.
Delete requires rD permission to the principal as well as d permission
to the principal directory.  Catalog requires r permission to the
principal directory.  Modify requires rfmu permission to the principal.
Rename requires rf permission to the principal.  Show requires r
permission to the principal.
dcecp>
```

- Get help about the syntax of an object by using an *object* **help -syntax** command. The following example displays the syntax diagram for the **principal** object:

```
dcecp> principal help -syntax
principal catalog [server_name_list] [-simplename]

principal create principal_name_list
           [-attribute attribute_list | attribute options]

principal delete principal_name_list

principal modify principal_name_list
           {[-change attribute_list | attribute options] |
           -add extended_registry_attribute_list |
           -remove extended_registry_attribute_list [-types]}

principal rename principal_name -to new_principal_name

principal show principal_name_list [-xattrs] [-all]

principal help [operation | -verbose] [-syntax]

principal operations
dcecp>
```

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*. Chapter 8, “ Writing Scripts and dcecp Objects” on page 69 explains the fundamentals of writing **dcecp** scripts and creating new objects.

Adding Scripts to dcecp Sessions

Note: In OS/390 UNIX System Services DCE, all **dcecp** script files must be in code page IBM-1047. If you port a script file from another platform, be sure that it is converted to code page IBM-1047.

After you have written a 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 called when **dcecp** initializes:

/opt/dcelocal/dcecp/init.dcecp

This file contains **dcecp**-specific startup information for the host. This affects all instances of **dcecp** running on a host. 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). The file is not shipped with DCE. Each DCE user can maintain a **.dcecprc** file and store his or her private procedures or alias names for operations. The file is used only when **dcecp** is started for interactive use, even when the commands are in a batch job.

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.

The rest of 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
1996-10-03-10:22:59.991-04:00I-----
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 has a user-created procedure called **show_clock**.

```
dcecp> show_clock
Time on mars is              1996-09-30-15:03:43.979-04:00I-----
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 run 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 run 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 calls 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>

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. This is in **/opt/dcelocal/dcecp**. As a rule, you should add the new object to each host in the DCE cell. Chapter 9, "DCE Administration Task Objects" on page 83 describes how you can use the **dcecp hostdata** object to copy scripts or other files to every host in a cell.

Note: In OS/390 UNIX System Services DCE, all **dcecp** script files must be in code page IBM-1047. If you port a script file from another platform, be sure that it is converted to code page IBM-1047.

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 Chapter 8, " Writing Scripts and dcecp Objects" on page 69.

You should also consider adding an invocation of the object procedure to **/opt/dcelocal/dcecp/init.dcecp** to make it known to **dcecp**.

Convenience Variables Mean Fewer Keystrokes

The **dcecp** program remembers what you type as well as command output, and 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 like **directory, principal, acl, account,** and so on. They do not apply to Tcl commands like **for** or **eval**, or UNIX commands like **mv** or **grep**. As an example, 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>
```

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 isn't 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.

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 **login** operation.

See the cell name variable description in “Current Cell Name (_c)” for information about composing fully qualified principal names.

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.

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

This variable is generally useful in environments where administrators deal with multiple cells. For example, 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>
```

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 started. Setting it using **set** generates an error.

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

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 **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>
```

Once set, you can use **$_n** in subsequent operations in place of the name argument. For example, 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>
```

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

Parent of the _n (_p)

The **_p** variable holds the parent of the name stored in **_n** or is set to the empty string if **_n** has no parent. The **_n** variable holds the name or list of names used in the argument to the most recent operation (see "Most Recent Operation Argument Name (_n)" on page 60). 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. For example the following operations view the ACLs of a server configuration object and of its parent object (**/.:/hosts/krypton/config/srvrconf**). Setting it using **set** generates an error.

```
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>
```

Last dcecp Object Name (_o)

The **_o** variable holds the name of the **dcecp** object used in the most recent operation. The following example uses the **_o** variable to avoid retyping **account**. Setting it using **set** generates an error.

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

Last Operation's Return Value (_r)

The **_r** variable holds the return value (such as a return code, output, or an error message) of the last processed interactive command. This variable is set only when **dcecp** is run in interactive mode. The variable may not be set by the user.

Notes:

1. Setting it using **set** generates an error.
2. Many **dcecp** commands return multiple lines of output which are in the form of a list.

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 **attrlist** in the *OS/390 DCE: 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>
```

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 master 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 started), 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 problem 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 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 problem 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. Usually, 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) /my_cell.goodco.com/hosts/planets/audit-server
/my_cell.goodco.com/hosts/planets/audit-server
dcecp>
```

When **_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 problem 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 cellname 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.

One reason to read the value of this variable is to check which registry performed the most recent operation as shown in the following example.

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

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. Refer to "DCE Servers to Use (_s(xxx))" on page 63 for information about the **_s(sec)** variable and how these values work together for registry operations.

The proc Command Lets You Create New Commands

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 Tcl **proc** command offers an easy way to create additional commands which look and act just like built-in Tcl commands such as **set, list**, and **while**. But 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>
```

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
EUVA11265E Cannot perform "read" on variable "x": EUVA08603E 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>
```

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 **unset** 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 Tcl **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 called 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 run scripts under the context of parent procedures using the Tcl **uplevel** command. This command offers a convenient way to manage your procedure's context. For instance, rather than import and manipulate numerous variables from a parent procedure, use **uplevel** to connect to them all at once. The syntax is:

```
uplevel ?level? arg ?arg ...?
```

The Tcl **uplevel** command is similar to **eval**; it concatenates arguments and runs them as scripts but unlike **eval**, **uplevel** runs 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
```

Running Operating System Commands from a Script

Although the DCE control program is versatile, there are times when you may want your script and executable files to use operating system commands to accomplish some simple (or even not-so-simple) operation. The Tcl **exec** command provides a way for scripts to perform external commands by creating a subprocess in which the command runs. 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 Tcl **exec** command usually returns the results of the operation performed in the subprocess. However, you can use UNIX-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**.

Note: Many non-**dcecp** commands can adversely affect the **dcecp** environment and should not be started from within **dcecp**. In particular, do not start **dce_login** (or DCELOGIN) to change the login context. Instead, use the **dcecp** built-in login command (see *OS/390 DCE: Command Reference* for more information). Also, do not attempt to start **dcecp** again from within **dcecp**.

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. 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 long_job &
dcecp>
```

Note: You do not have to specify **exec** if the command name does not conflict with the name of a **dcecp** or Tcl command or procedure.

Chapter 8. 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 act 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 multi-component 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. This chapter provides information for extending, modifying, or creating the following kinds of dcecp scripts:

- informal administration scripts
- formal task objects

Notes:

1. This chapter assumes that you know enough about the Tcl language to create your own scripts. To learn how to use Tcl, consult a book on the subject, such as *Tcl and the Tk Toolkit*, by John K. Osterhout, (c)1994, Addison-Wesley Publishing Company.
2. In OS/390 UNIX System Services DCE, all **dcecp** script files must be in code page IBM-1047. If you port a script file from another platform, be sure that it is converted to code page IBM-1047.

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.

Scripts generally consist of one or more procedures created with the **proc** command. This lets you run 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 56. 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. For instance you could rename the 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>
```

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.

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."]
            }
        }
    }
}
```

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.

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 run.
- 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 runs.

The rest of this section shows the general structures and conventions used in a formal task object. To aid the explanation, the **dcecp user** task object supplied with the DCE control program is used.

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 act 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, some repetitive parts of the included procedures have been omitted; the omitted parts were replaced with vertical ellipses in the code examples. (We have also omitted debug statements.) The entire **user** task object is contained in the **/opt/dcelocal/dcecp/user.dcp** script file.

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 entities 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. The **parseargs** procedure is explained in “Using the parseargs Procedure” on page 78.

```
# 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
       "Creates a DCE user." }
    {delete command function_call _dcp_delete_user
       "Deletes a DCE user."}
    {show   command function_call _dcp_show_user
       "Shows the attributes of a DCE user."}
    {help   help  help_list
       "Prints a summary of command-line options."}
    {operations operations operation_list
       "Returns the valid operations for command."}}

set syn_table {
  {create_syntax "user create user_name -mypwd password \
     -password password -group group_name -organization \
     organization_name [ -force ]"}
  {delete_syntax "user delete user_name"}
  {show_syntax "user show user_name"}
  {help_syntax "user help [ operation | -verbose | -syntax ]"}
  {operations_syntax "user operations"}

  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 $syn_table local_args -found_one -no_left
  if { [llength [info local help_prose ]] > 0 } { return $help_prose }
  if { [llength [info local function_call ]] > 0 } {
    return [$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.

```
  .
  . several low-level procedures omitted
  .

#
# This proc 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
      principal create $principal_name
  } else {
    error "Principal \"$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 account's principal have duplicate keys."}
    {-expdate string expdate
           "When does the account expire."}
 .
 .  other elements omitted
 .
    {-uid integer uid
         "User Identifier of the principal to be added."}}

  set syn_table {
  }

#
# 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 $syn_table local_args -no_leftovers

    if { [llength [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 { [llength [info local attribute_list]] > 0} {
       set pile_of_attributes "alias client descr dupkey expdate\
       forwardabletkt 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 { [llength [info local group]] > 0} {
      set account_args [format "%s {%s %s}" $account_args group $group]
    } else { error "No group name specified." }

    if { [llength [info local organization]] > 0} {
      set account_args [format "%s {%s %s}" $account_args organiz $organization]
    } else { error "No organization name specified." }

    if { [llength [info local password]] > 0} {
      set account_args [format "%s {%s %s}" $account_args password $password]
    } else { error "No password specified." }

    if { [llength [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 { [llength [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 { [llength [info local uid]] > 0 } {
      set princ_args [format "%s {%s %s}" $princ_args uid $uid]
    }
  .
  .  other elements omitted
  .

    if { [llength [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 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

    set syn_table {
    }

    parseargs "" $syn_table local_args -no_leftovers

    if { [llength [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 syn_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
```

and

```
-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**, **help**, and **operations**.

syn_options: the syntax table describing the syntax of the parse options. There must be a syntax entry for each verb entry in the parse options. The form of the syntax table is:

syntax_verb	a name which corresponds one-to-one with the command name.
syntax_string	the syntax for the command.

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 called within the script.

args: one or more of three 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 more than one is found. This is used on second level command parsing to ensure the user entered at most one argument in the command.

-no_left
generates an error if the first argument is not a verb in parse_options. This is used on initial command parsing to ensure the user entered a known verb for the object.

Invoking 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 run it as described in "Informal Administration Scripts" on page 69.

Formal task objects require a few steps to make them act 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 in a file named **tclIndex** that informs the DCE control program about the procedures in a directory. 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>
   ```

   Note: In OS/390 UNIX System Services DCE, all **dcecp** script files must be in code page IBM-1047. If, after you run **auto_mkindex**, the **tclIndex** file is not in code page IBM-1047, use **iconv** to convert it.
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.
4. To make the new task object known to **dcecp** during initialization, add the following record to **/opt/dcelocal/dcecp/init.dcecp**.

   ```
   catch object_name
   ```

You need to make this file available on each DCE host where the script will be run. 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 post-processor attribute.

Part 4. DCE Administration Tasks

Chapter 9. DCE Administration Task Objects 83
Using Task Objects to Simplify DCE Administration 83
Getting Started with Objects 84
Looking Beyond the Tools 84

Chapter 10. Managing a DCE Cell 85
Showing All Configured DCE Servers and DCE Hosts 85
Testing Cell Operation 86
Modifying or Extending the Cell Object 87
| Using the eNetwork Communications Server with OS/390 DCE 87

Chapter 11. Managing DCE Hosts 89
Listing the DCE Hosts in a Cell 89
Showing All Servers Configured for a DCE Host 90
Testing Whether a DCE Host Is Running . . . 90
Starting Configured DCE Processes on a Host 91
Stopping DCE Processes Running on a Host . 91
Modifying or Extending the Host Object 91

Chapter 12. Managing DCE Users 93
Creating a New User 93
Showing User Information 94
Deleting a User 95
Modifying or Extending the User Object 96

Chapter 9. DCE Administration Task Objects

This part of the book discusses the purpose and use of DCE administration task objects provided with OS/390 DCE Release 1. Generally, these special **dcecp** objects perform routine high-level administration tasks by combining several lower-level operations.

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.

While the task objects are discussed at a high level, you will need to keep in mind that there is often more going on that is only hinted at. In these cases, you will be told where to go in this guide for more detailed information. Usually you will be directed to the corresponding lower-level discussion in the relevant component's part of this guide.

Using Task Objects to Simplify 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 lets you 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. As an example, let's 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 wouldn't 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, the term *task* is used to mean doing something that requires multiple steps, such as when adding a user consists of performing six lower-level operations.

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. See Part 3, " The DCE Control Program" for how to use the DCE control program language to write and modify a **dcecp** task object.

Getting Started with Objects

Online help for an object is available using the *object_name* **help** and *object_name* **operations** commands (where *object_name* is the name of the object) in **dcecp**.

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

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 are managing. Each of the 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.

The remaining chapters in this part discusses how to manage these DCE elements by using the default implementations of the **dcecp** task objects provided with OS/390 DCE Release 1.

Chapter 10. 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, doesn't 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.

Showing All Configured DCE Servers and DCE 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.

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 hasn't grown accordingly. Or perhaps you've just been hired to administer a new cell and you want to see what your cell consists of.

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. 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.

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/bigbox}
{dtsservers
 /.../my_cell.goodco.com/hosts/duh}
{hosts
/.../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>
```

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>
```

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.

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**, with each hostname as an argument.

In case of a problem, the operation generates an error message and returns a list of servers or hosts that could not be contacted. For complete success with the **-replicas** option, the operation returns the message **DCE servers available**. For complete success with the **-clients** option, the message is **DCE clients available**. If you specify both options, the complete success message is **DCE clients and servers available**.

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>
```

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.

Part 3, " The DCE Control Program" discusses ways to create new **dcecp** objects or modify existing objects written with the **dcecp** language.

Using the eNetwork Communications Server with OS/390 DCE

In the past, High Performance Data Transfer (HPDT) for UDP and High Speed Access Services (HSAS) were TCP/IP implementations that were tightly coupled with the OS/390 UNIX System Services kernel. These products were designed to achieve higher performance and reduce CPU utilization compared to earlier OS/390 TCP/IP implementations. Now, starting with OS/390 Release 6, the features provided by HPDT and HSAS are now available in the base TCP/IP Services component of the eNetwork Communications Server for OS/390.

In all three releases of the TCP/IP Services (HPDT, HSAS, and the eNetwork Communications Server for OS/390), there are some DCE-related considerations regarding:

- The use of the Enterprise System Connection (ESCON®)
- The use of the environment variable _BPXK_INT_FASTPATH.

Servers, including those that are part of the DCE Kernel, may register ESCON addresses in the local endpoint map, and place them in the name service database for access by any client. DCE planners should insure that these addresses are reachable by all hosts in the DCE cell that wish to use a server that advertises these addresses. This type of connection is point-to-point between the OS/390 system and an AIX® system. If servers are allowed to register these addresses in the local endpoint map and with the name service database, many hosts in the DCE cell may not have connectivity to these point-to-point addresses. If this is the case, the environment variables RPC_UNSUPPORTED_NETADDRS or RPC_UNSUPPORTED_NETIFS can be used to prevent servers from using these addresses.

DCE does not support the use of the fast path environment variable _BPXK_INT_FASTPATH. Applications are not XPG4-compliant while using this environment variable, in that POSIX signals are not supported. DCE Remote Procedure Call is dependent on POSIX signals and therefore does not operate correctly when the fast path environment variable is enabled.

Chapter 11. 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.

Listing the DCE Hosts in a Cell

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 doesn't 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 Chapter 25, "Managing CDS Directories" on page 219.

The **host catalog** operation resembles the **cell show** operation except it doesn't separately list DCE servers. 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 cellname 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>
```

Showing All Servers Configured for a DCE Host

In larger cells, in which DCE servers and application servers reside on multiple hosts, you may 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.

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 Chapter 14, "DCE Application Administration" on page 105.

The following example shows the servers configured to run on DCE host **bigbox**:

```
dcecp> host show /.:/hosts/bigbox
{video_clip running}
dcecp>
```

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 isn't accessible.

The **host ping** operation tests for the presence of the remote host's DCE Host daemon (**dced**). You can read more about the purpose and use of the **dced** in Chapter 13, "Managing DCE Host Services and Host Data" on page 99.

The following example tests whether the **dced** on host **duh** is accessible on the network.

```
dcecp> host ping /.:/hosts/duh
1
dcecp>
```

Starting Configured DCE Processes on a Host

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 lets you start all configured servers on a host with a single command. On non-OS/390 based hosts, the **host start** operation may also allow you to start all configured DCE servers and clients. See Chapter 5, "Starting and Stopping OS/390 DCE" on page 33 for information on starting and stopping DCE daemons.

To operate on a remote host, its DCE Host 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 all servers that are configured by using the **server** object. DCE servers and clients on OS/390 hosts are not configured as server objects. Application servers must be configured with the **starton** attribute set to **boot** or **explicit**. You can read more about configuring application servers in Chapter 14, "DCE Application Administration" on page 105.

The following example starts all configured servers on host **bigbox**:

```
dcecp> host start /.:/hosts/bigbox
dcecp>
```

Stopping DCE Processes Running on a Host

Like the **host start** operation discussed in the previous section, the **host stop** operation is more encompassing than a **server stop** operation. It lets you stop all application servers and DCE processes on a non-OS/390 based host with a single command rather than enter 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**. On an OS/390 based host, only configured application servers can be stopped with the **host stop** operation. See Chapter 5, "Starting and Stopping OS/390 DCE" on page 33 for information on starting and stopping DCE daemons. You can read more about controlling servers in Chapter 14, "DCE Application Administration" on page 105.

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 all servers that are configured by using the **server** object.

The following example stops all DCE processes and application servers on host **bigbox**:

```
dcecp> host stop /.:/hosts/bigbox
dcecp>
```

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.

Part 3, " The DCE Control Program" of this guide discusses ways to create new **dcecp** objects or modify existing objects written with the **dcecp** language.

Chapter 12. 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. 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.

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.

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 run 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 section of the *OS/390 DCE: Command Reference*.

   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 non-alphanumeric characters. You can read more about administering principals, groups, and organizations in Chapter 36, “Creating and Maintaining Principals, Groups, and Organizations” on page 331.

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 section of the *OS/390 DCE: Command Reference*.

   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 Chapter 37, “Creating and Maintaining Accounts” on page 347.

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

Chapter 25, “Managing CDS Directories” on page 219. You can read more about ACLs and CDS directories in Chapter 23, “Controlling Access to CDS Names” on page 197.

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** section of the *OS/390 DCE: Command Reference* lists the required permissions.

To create a new user in a DCE cell, run 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 these 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>
```

Note: When you add new user accounts, and one or more of those users is to be cross linked to a new RACF ID, remember to run the RACF interoperability utility, **mvsexpt**, so that the new users will have single sign-on capability and interoperability between RACF and OS/390 DCE. For more information, see “Cross Linking Existing DCE Users who are New RACF Users” on page 405.

If the new user accounts are to be cross-linked to existing OS/390 RACF IDs, you may want to use the RACF interoperability utility, **mvsimpt**, to create the DCE users in the DCE registry instead of manually invoking the **dcecp** command. You can then run **mvsexpt** to create the RACF DCE segment for the OS/390 RACF IDs. For more information, see “Cross Linking Existing RACF Users who are New DCE Users” on page 400.

Also, for users who are to be enabled for single sign-on, remind them that they must each use the OS/390 DCE **storepw** command before invoking a DCE application from the OS/390 system. The **storepw** command must also be run any time the user changes his or her password. For more information on the **storepw** command, see the *OS/390 DCE: Command Reference*.

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:

1. **principal show**
2. **account show -all**

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}
EUVA04672I No policy.
dcecp>
```

You can show information about multiple users by specifying a list of user names as an argument to the **user show** operation.

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.

You need numerous permissions, such as those generally associated with cell administrator, to delete a user. See the **user** object section in the *OS/390 DCE: Command Reference*.

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>
```

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.

Part 3, " The DCE Control Program" on page 43 discusses ways to create new **dcecp** objects or modify existing objects written with the **dcecp** language.

Part 5. DCE Host and Application Administration

Chapter 13. Managing DCE Host Services and Host Data 99
DCE Host Services 99
  Starting and Stopping DCE Host Services .100
Managing Host Data 100
  Permissions For Accessing Host Data . . 101
  Modifying Host Cell Name Information . . 102
  Manipulating Data in Other Host Files . . . 103

Chapter 14. DCE Application Administration 105
Controlling Server Operation 105
  Common Server Configuration Needs . . . 106
  Configuring Servers 111
  Listing and Retrieving Server Configuration Information 112
  Unconfiguring Servers 112
  Starting and Stopping Servers 112
  Disabling and Enabling Services 113
  Extending Server Configurations 113
  Changing Server Configurations 115
  Checking Whether Servers Are Running . 116
Managing Client-Server Binding Information . 116
The Endpoint Map Eases Application Development and Administration 118
  The Endpoint Map 118
  Endpoint Map Administration is Mostly Automatic 118
  Commands for Monitoring the Endpoint Map 119
  Commands for Rebuilding the Endpoint Map 119
  Command to Determine if a Server is Listening 120
Other Endpoint Map Administration Tasks . . 120
  Deleting the Endpoint Map During Startup .120
  Recovering the Endpoint Map 120
  Restricting Endpoints 121
  Viewing Information in the Endpoint Map . 122
Managing Server Entries, Groups, and Profiles in CDS 123
  The RPC_DEFAULT_ENTRY Environment Variable 123
  Unique Server Entry Names Identify Individual Servers and Objects 123
  Group Entries Help Balance Server Workloads 127
  Profiles Help Direct Clients' Searches For Servers 130
Client Administration 135
  Determining the Entry Name 136
  Providing the Entry Name to Clients 136

Chapter 15. Examples of Setting Up RPC Profiles 139
Math Server Example 139
  Frequency of Use 139
  Solution 140
  Steps in Creating the RPC Profile 140
Print Server Example 142
  Frequency of Use 143
  Solution 143
  Steps in Creating the RPC Profile 143

Chapter 16. Controlling Access to the DCED Endpoint Map 149
Overview of DCED Endpoint Map Authorization 149
  ACL Entry Types Supported by the DCE Endpoint Map ACL Manager 149
  Permissions Used for the DCE Daemon Endpoint Map 149
DCE Clients' Access to DCED 150
Giving Application Servers Access to the DCED Endpoint Map 150
  RPC Server Group 150
Giving DCE Administrators Access to the DCED Endpoint Map 151
The Default Endpoint Map ACL 152
Giving Unauthenticated Users Access to the DCED Endpoint Map 152

Chapter 17. Workload Balancing in a Parallel Sysplex Environment 153
Overview of Workload Balancing 153
Setting Up Workload Balancing 154
  Server Requirements 154
  Client Requirements 154
Using Workload Balancing 154
  Using the Environment Variable 154
  Using the Application Programming Interfaces 155
  Using dcecp endpoint Commands with Workload Balancing 155
  The PI Program 155

When you first think about administration tasks in a DCE environment, it is natural to think about the large, cell-wide services like the Cell Directory Service or the DCE Security Service because of their scope and complexity. But a closer look at what constitutes a DCE cell reveals other units of operation and administration.

The individual host computer is one obvious unit of operation and administration. After all, it is the individual host computers that support user logins, run DCE software, and run DCE-based client/server applications. Host computers have principal identities and cell affiliations that must be maintained. Chapter 13, “Managing DCE Host Services and Host Data” on page 99 discusses DCE host administration.

The DCE-based client/server application is another unit of operation and administration. While DCE applications certainly have their own application-specific administration needs, they also rely on a set of commonly used DCE services that support distributed operation. Application administrators might need to manage an application's use of these common DCE services. Chapter 14, “DCE Application Administration” on page 105 discusses DCE distributed application administrative tasks.

Chapter 13. 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 chapter.

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. The second part of this chapter discusses how to use the DCE control program to gain remote, authenticated access to this data.

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 OS/390 DCE Release 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. See “The MODIFY DCEKERN Operator Command” on page 35 for information on starting daemons.

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.

Usually, 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's 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 problem in 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

Before starting **dced**, any underlying network services on which client/server communication depends must be available; on most systems, for example, network interfaces and routing services must be enabled. After these transport-layer services are established, you can start DCEKERN, which starts **dced**. After **dced** starts, RPC-based servers are started.

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 is not 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 IPLs because most servers get different endpoints and different process id's each time they start.

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.

If the DCE daemon stops or exits unexpectedly, it is usually restarted automatically. However, you can restart it manually, if necessary. The restarted **dced** does *not* lose any previously registered server bindings. It simply loads the information from the **Ep.db** file. As a rule, stopping and restarting the **dced** is not recommended because it also stops the security validation service.

After you've started the DCE host services, you can perform most 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 most of your OS/390 DCE Release 1 hosts without having to log in to each host. In Part 3, “ The DCE Control Program,” how to use **dcecp** in interactive mode, as well as how to write **dcecp** scripts to manage DCE activities, are explained. It helps to understand those basics before performing administrative tasks explained in this book.

Managing Host Data

Each host in a DCE cell maintains local data that is essential for operating in a DCE environment. For instance, each host's DCE identity relies on certain data items that specify the host's hostname, cell name, and any cell aliases. 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. See “Modifying Host Cell Name Information” on page 102 for additional information and cautions on this usage.

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. The last part of this chapter examines this powerful access method.

Permissions For Accessing Host Data

Access control lists (ACLs) prevent unauthorized principals from creating, changing, or deleting hostdata information. Two types of ACLs protect hostdata information. One type of ACL protects the container in which the hostdata items reside. A second type protects each individual hostdata item.

This section shows how to manage ACLs that protect hostdata information. For detailed information about setting and using ACL protections, read Chapter 34, "Using Access Control Lists" on page 307.

Permissions for the Hostdata Container: In OS/390 DCE Release 1, the hostdata items reside in a *container* which is really a backing storage mechanism maintained by **dced**. On most 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 hostdata container ACL with the following name:

/.../*cellname***/hosts/***hostname***/config/hostdata**

The hostdata container ACL has the following permissions:

c (control)	Modify the container ACL
r (read)	Read the list of hostdata items in the container
i (insert)	Create new hostdata items
I (Insert)	Although the **I** permission is present, it does not apply to hostdata items. The permission applies to server control facilities which are explained in Chapter 14, "DCE Application Administration" on page 105.

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>
```

Permissions for the Hostdata 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

Use the **acl** object to view or modify ACLs. For example, use the following operation to view the ACL for the **cell_aliases** hostdata item on host **silver**.

```
dcecp> acl show /.:/hosts/silver/config/hostdata/cell_aliases
{user hosts/silver/self cdprw}
{unauthenticated r}
{any_other r}
```

Modifying Host Cell Name Information

Using the **hostdata** object, you can add, change, and remove data items on DCE hosts. Each DCE host maintains a protected local copy of the cell name 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**. Each host uses 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.

When such a situation 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. On OS/390, this is done by using DCECONF to deconfigure and reconfigure a host.

Note though, that 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.

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 cellname to be **/.../my_cell.goodco.com** on host **silver**.

```
dcecp> hostdata cat
/.../my_cell.goodco.com/silver/config/hostdata/dce_cf.db
/.../my_cell.goodco.com/silver/config/hostdata/cell_name
/.../my_cell.goodco.com/silver/config/hostdata/pe_site
/.../my_cell.goodco.com/silver/config/hostdata/host_name
/.../my_cell.goodco.com/silver/config/hostdata/cell_aliases
/.../my_cell.goodco.com/silver/config/hostdata/post_processors
dcecp> hostdata show cell_name
{uuid 00174f6c-6eca-1d6a-bf90-0000c09ce054}
{annotation  {Name of cell}}
{storage cell_name}
{hostdata/data /.../old_cell.goodco.com}
dcecp> hostdata modify /.../my_cell.goodco.com/hosts/silver/config/hostdata/cell_name \
>-data {/.../my_cell.goodco.com}}
dcecp>
```

If DCEKERN requires a code page other than IBM-1047, use the UNIX System Services **iconv** command to convert **opt/dcelocal/bin/dcecf_postproc** to the new code page before starting DCEKERN. As initially installed, **dcecf_postproc** (a postprocessor shell script) is in the IBM-1047 code page. See the *OS/390 UNIX System Services Command Reference*, SC28-1892, for information on the **iconv** command. See the *OS/390 DCE: Application Development Guide: Core Components* for additional information on managing host data.

Manipulating Data in Other Host Files

While the **hostdata** object is useful for changing cell name information, it has a broader use too; you can use it to add, change, and remove data from any HFS file that is accessible on a DCE host.

One useful example is adding a new CDS attribute. Every DCE host has its 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-runs 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>
   ```

Chapter 14. DCE Application Administration

As DCE evolves, commonly needed functions are being included in the DCE infrastructure. As an example, OS/390 DCE Release 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. Some programmers can be expected 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. Discussions in this chapter will include this area of overlap, noting circumstances where administrative action might be needed.

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 Host 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.

The following sections show some common configuration needs and describe ways to configure and unconfigure servers, how to start and stop servers, and how to view server information.

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 Chapter 37, "Creating and Maintaining Accounts" on page 347.

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). The following example shows configuration information for the **video_clip** server on host **krypton** in remote cell **/.../my_cell.goodco.com**:

```
dcecp> server show /.../my_cell.goodco.com/hosts/krypton/config/srvrconf/video_clip
{uuid 2fa417e8-bb4c-11cd-831b-0000c08adf56}
{program {vclip}}
{arguments {-catalog}}
 .
 . (Output Omitted)
 .
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}}
 .
 . (Output Omitted)
 .
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}}
 .
 . (Output Omitted)
 .
dcecp>
```

Server Configuration Information: Each OS/390 DCE Release 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 that calls the server program. In OS/390 DCE, if the program is a script, it must have #! as the first two characters in the file. They should be entered using the same codepage that DCEKERN uses when the server is started. Then **/bin/sh** is called to run the script. At this time, **stdin**, **stdout**, and **stderr** are not open, so the server program must

open them as necessary, for example by using redirection in a shell script to route output to an HFS file.

directory The name of the program's working directory. After a server is running, it might need a place to store its output or temporary files. In OS/390 DCE, if a directory is not specified, it defaults to **/tmp**.

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 comment field limited to Portable Character Set (PCS) data that cannot be modified after creation.

- **ifname** The interface name of this service, limited to PCS characters and specified in the interface definition file.

- **interface** The interface identifier (UUID and version number) of this service (specified in the interface definition file).

- **bindings** A list of string bindings identifying this service. This is specified only for well-known endpoints.

- **entryname** The name of an RPC entry to which the server exports its binding for this service (limited to PCS characters).

- **flags** A list of keywords to identify flags for this server. Only the **disabled** flag is currently supported. This flag indicates the mapping was disabled in the endpoint map.

- **objects** A list of object UUIDs supported by this service.

uid A POSIX uid that the server is started with.

gid A POSIX group ID 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. This function is not currently supported.

- **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 **dced** is started.

- **failure** The server is restarted whenever it has exited with a non-successful exit status.

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 Chapter 34, “Using Access Control Lists” on page 307.

Permissions for the Server Configuration Container

The server configuration information resides in a **container**. The container, a backing storage mechanism implemented as a file, 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.

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**.

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 (run)	Start server

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>
```

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 for stopping 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, 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

I\ (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.

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**.

Permissions For Accessing Server Execution Information

Each server's execution information is protected by its own ACL. These ACLs can prevent unauthorized principals from creating, changing, or reading server execution information, and from stopping 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

As an example, 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>
```

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 **dced** starts 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 when **dced** is started.
Auto	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. This function is not currently supported.
Failure	You can set a server to start automatically if it exits unexpectedly.

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** calls the server which is located in the **/usr/local/bin** working directory. The server has a catalog mode that is 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 **dced** is started. 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>
```

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 doesn't 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.

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>
```

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. For example, 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 using the **hard** method.

```
dcecp> server stop video_clip -method hard
dcecp>
```

Disabling and Enabling Services

Note: These functions are not supported when the target DCE server is running on an OS/390 host.

You can prevent clients from using a service offered by a server—even when the server is running—by 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. 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>
```

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.

For example, say 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>
```

After you have created a new attribute, use a **server modify** operation as explained in the section "Changing Server Configurations" on page 115 to insert the necessary data. More information about ERAs is provided in Chapter 38, "Creating and Using Extended Registry Attributes" on page 363.

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>
```

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
I (Insert)	Although the **I** permission is present, it does not apply to ERA items. The permission applies to server control facilities which are explained in “Permissions for Accessing Server Control Facilities” on page 108.

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>
```

The ACL for an individual ERA is named:

/.../*cellname***/hosts/***hostname***/config/xattrschema/srvrconf/***ERA_name*

The ACL for an individual ERA has the following permissions:

c (control)	Modify the ACL
r (read)	Read ERA in the container
w (write)	Modify the ERA information
d (delete)	Delete the ERA information

ACLs on individual ERAs can prevent unauthorized principals from creating, reading, changing, or deleting ERA information. The following example shows permissions established for the **objfamily** ERA. 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 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 **xattrschema** container on host **silver**:

```
dcecp> acl show /.:/hosts/silver/config/xattrschema -io
{unauthenticated r}
{any_other r}
dcecp>
```

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 usual 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 steps are illustrated in the following example with 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 {boot}}
{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 \
>-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” on page 113.

```
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** is a server is listening and a **0** if it is not listening. The following example tests whether the **video_clip** server is running.

```
dcecp> server ping /.:/hosts/silver/config/srvrconf/video_clip
1
dcecp>
```

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.

Clients need three pieces of information to communicate with a server:

- The hostname (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 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 hostname. 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 9 on page 117 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**.

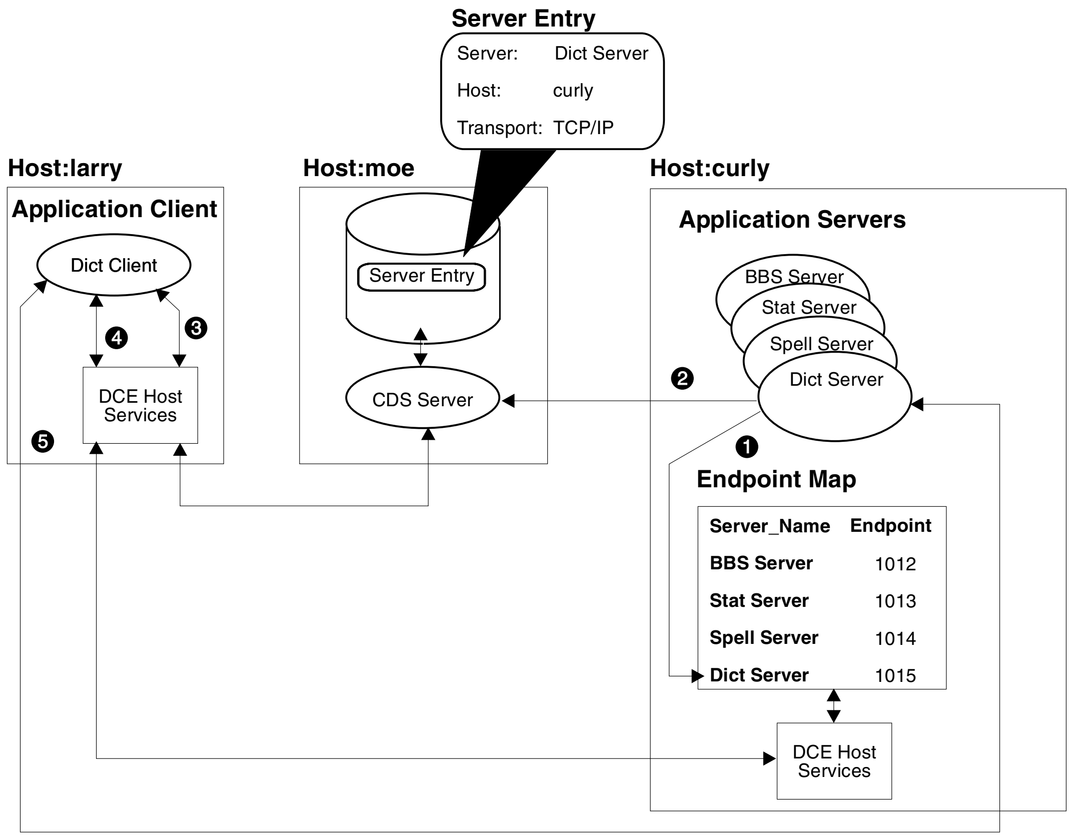


Figure 9. Server Binding Information

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 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 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 some details in this high-level example have been omitted, 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 described. But these alternatives are generally controlled by the applications themselves rather than through conventional DCE administration facilities like **dcecp**.

This section discussed one basic client/server binding mechanism. The following sections examine the roles played by the endpoint map and by CDS. We'll also discuss specific administration tasks for managing binding information in endpoint maps and in CDS.

The Endpoint Map Eases Application Development and Administration

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.

While well-known endpoints provide convenient access to some critical servers, for most servers they are impractical. That's 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 needn't worry about passing dynamic endpoints to clients. Furthermore, unless a server moves to a new host, or removes or adds a transport, it doesn't even have to update the information in CDS.

The Endpoint Map

The Endpoint Map Service maintains RPC server information in a database called the **endpoint map**. Each server entry in this database includes fully specified binding information for the server, UUIDs for objects and interfaces offered by the server, an annotation string, and other fields. A server registers with the local endpoint map using either the **rpc_ep_register** or **rpc_ep_register_no_replace** programming interface, or the **dce_server_register** programming interface.

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 DCEKERN startup, 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 if the **rc** file was modified to allow this. See "Deleting the Endpoint Map During Startup" on page 120 for details. The **Ep.db** file is also deleted each time DCEKERN is restarted.

DCE-based servers usually 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. OS/390 DCE Release 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 (through the **rpc_ep_unregister** or the **dce_server_unregister** programming interface) when they no longer service RPCs. Rather, this processing is intended only to clean up after a server problem.

When the DCE daemons on non-OS/390 hosts are recycled, they register new entries in the endpoint map (containing new endpoints). However, the old (and stale) entries of these daemons are not always deleted from the endpoint map. If the OS/390 DCE host interacts with such hosts, the OS/390 DCE host may not be able to perform any DCE tasks. In this case, make sure that the stale endpoint map entries in the endpoint map of the non-OS/390 host are purged.

On OS/390, servers that register with the endpoint map are monitored through OS/390 facilities. An abnormal end of the server task or its address space causes the server's endpoints to be removed from the endpoint map.

Commands for Monitoring the Endpoint Map

OS/390 DCE provides additional **rpccp** commands you can use to monitor the status of the endpoint map. To query the status of the endpoint map, use the **query epdb** subcommand. To determine the status of the endpoint map, enter the following from the **rpccp** prompt:

```
rpccp> query epdb host-address
```

The *host-address* parameter is the combination of the network address (Internet address) and the protocol sequence of the host where the endpoint map is located. The *host-address* parameter is optional when querying the local host. For example, to query the status of the endpoint map on the local host, enter:

```
rpccp> query epdb
```

To query the status of the endpoint map on a remote OS/390 host whose Internet address is 9.21.21.91, enter:

```
rpccp> query epdb ncadg_ip_udp:9.21.21.91
```

The display from this command will look similar to the following:

```
Endpoint database file exists.
Total number of active registered entries = 10
```

Commands for Rebuilding the Endpoint Map

To rebuild the endpoint map, use the **rebuild epdb** subcommand of **rpccp**. It rebuilds the endpoint map from the RPCD in-memory entries.

First delete the file **/opt/dcelocal/var/dced/Ep.db**. (You must have the proper authority to do this.) You must also have **c** (control) permission to the endpoint map object to be able to run the **rebuild** command successfully.

The syntax of this command is:

```
rpccp> rebuild epdb host-address
```

where *host-address* is the combination of the protocol sequence and the network address (Internet address) of the host that has the endpoint map. This parameter is optional when you are rebuilding the endpoint map on the local host. For example, to rebuild the endpoint map on the local host, enter:

```
rpccp> rebuild epdb
```

To rebuild the endpoint map on a remote OS/390 host whose Internet address is 9.21.21.91, enter:

```
rpccp> rebuild epdb ncadg_ip_udp:9.21.21.91
```

Command to Determine if a Server is Listening

Use the **ping** subcommand of RPCCP to determine if a server (DCE servers or user-written applications) is accessible from the OS/390 host system. To use this command, you need the complete string binding of the server, for example:

```
rpccp> ping ncadg_ip_udp:9.21.22.196[135]
```

Use the show mapping command to obtain the complete binding information of a server. Optionally, you can specify the number of times the server is to be pinged, through the **count** parameter. If the server is accessible, a message similar to the following is displayed:

```
Ping #1 response took 200 msec
```

This message is displayed for each successful ping, and is repeated according to the number specified in the count parameter.

If the server is not accessible, a message similar to the following is displayed:

```
Ping #1 timed out after 30000 msec
```

Other Endpoint Map Administration Tasks

OS/390 has additional ways to aid in endpoint map administration.

Deleting the Endpoint Map During Startup

An existing endpoint map file must be deleted before starting up the DCEKERN address space. If an old endpoint map file exists, the DCE daemon may not be able to start up correctly.

You can remove the endpoint map file, **/opt/dcelocal/var/dced/Srvrexec.db** during the startup of OMVS by including an **rm /opt/dcelocal/var/dced/Srvrexec.db** command in the **/etc/rc** file. This file is called during OS/390 initialization. The other endpoint map file, **/opt/dcelocal/var/dced/Ep.db** is automatically deleted and re-allocated when DCEKERN is started.

Recovering the Endpoint Map

OS/390 DCE provides recovery mechanisms if the endpoint map is corrupted. This section describes three different recovery scenarios.

Recovering from I/O Error During Normal Operation: At any time, there are two existing copies of the endpoint map: the physical file that resides on disk and the temporary in-memory copy. If DCED cannot update the physical endpoint map, DCED makes it unavailable to **QDCEADMIN/EUVRPC(RPCDEPBAK)** so that it cannot be used. All subsequent attempts by servers to register their endpoints in the endpoint map will be rejected by DCED. However, servers can continue to unregister their endpoints in the in-memory copy of the endpoint map. DCED will also log the error in a log file. Because the in-memory copy of the endpoint map is still valid, you can **rebuild** the physical endpoint map using the in-memory copy.

After you have resolved all the I/O problems that caused this error, you can rebuild the physical endpoint map using the **rebuild epdb** subcommand of the RPC Control Program. Details on using this command can be found in "Commands for Rebuilding the Endpoint Map" on page 119.

Recovering from Endpoint Map I/O Errors: DCED may also end abnormally immediately after an I/O error. If this is the case, and if the endpoint map has been corrupted, delete it and restart DCED. DCED will create a new endpoint map, and all servers will have to register with DCED again.

Recovering a Corrupted Endpoint Map at DCED Startup If the endpoint map was corrupted while DCED was not running, it must be removed and the entire DCE subsystem has to be restarted. All DCE servers as well as application servers will have to register with DCED again.

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.

| UNIX System Services provides the INADDRANYPORT and INADDRANYCOUNT parameters to control
| the dynamic assignment of ports. The range of ports specified for the environment variable
| RPC_RESTRICTED_PORTS must not intersect with those specified by INADDRANYPORT or
| INADDRANYCOUNT. See *OS/390 UNIX System Services Planning*, SC28-1890, for information about
| INADDRANYPORT and INADDRANYCOUNT. Due to performance considerations, use of
| INADDRANYPORT or INADDRANYCOUNT is preferred over RPC_RESTRICTED_PORTS.

| The TCP/IP PORTRANGE statement may also affect port usage. See *OS/390 eNetwork Communications
| Server: IP Configuration*, SC31-8513, for information. The use of ports specified by
| RPC_RESTRICTED_PORTS, INADDRANYPORT, or INADDRANYCOUNT must not be prohibited by
| PORTRANGE.

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 ports by the RPC runtime. It does not affect well-known endpoints.

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.

```
$ export 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 CDSADV, set the environment variable each time before starting your cell.

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.

Note: If a host has more than one IP interface (that is, it has multiple Internet addresses), and the networks for the different interfaces are not connected (for example, through routers) to each other, a problem may be encountered when a DCE client attempts to request a service from a server on this host.

When an application server on this host starts up, it registers multiple binding information based on these Internet addresses. As such, when a DCE client makes a call to this server, it uses one of these bindings. This call can only be successful if the client used the binding information based on the Internet address corresponding to the interface that is connected to the network of the DCE client host.

In this case, the network administrator must ensure that DCE clients can connect to all network interfaces on a server machine.

Viewing 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.

On OS/390, endpoints are protected by ACLs. Anyone who can run **dcecp** can use the **endpoint show** command 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 servers or users with the appropriate permissions on the local host, or by users with the appropriate permissions on a remote host. See Chapter 16, “Controlling Access to the DCED Endpoint Map” on page 149 for more information.

You can view information stored in a host's endpoint map database by using an **endpoint show** operation. The following example shows the endpoint map information for the **video_clip** server on a remote host **9.21.22.196**, using the datagram protocol. Omit the hostname argument to operate on the local endpoint map.

```
dcecp> endpoint show ncadg_ip_udp:9.21.22.196 \
-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>
```

You can view all of the endpoints in an endpoint map by not using any options with the **endpoint show** operation.

Managing Server Entries, Groups, and Profiles in CDS

An endpoint map acts as a directory of servers on a host. Similarly, CDS acts as a directory of servers in the cell. In the first part of this Chapter, we gave a high-level look at how applications can use CDS to store relatively stable binding information such as a server's name, its host address and the transports over which the server is available. In this section, how to use CDS facilities for organizing your servers and other distributed objects in meaningful ways is shown.

Many of the operations discussed in the following sections operate on CDS directories which are protected against unauthorized access by ACLs (access control lists). For detailed information about ACLs and CDS see Chapter 23, “Controlling Access to CDS Names” on page 197.

The RPC_DEFAULT_ENTRY Environment Variable

When a client searches for binding information, it can use a specific entry name. Alternatively, it can use a default starting point that is set as an environment variable. The RPC_DEFAULT_ENTRY environment variable sets the default starting point by which a client performs its search for compatible bindings. You can set this environment variable to a server entry name, a group name, or the name of a profile. Setting environment variables is described in Appendix A, “Environment Variables in OS/390 DCE” on page 467.

Unique Server Entry Names Identify Individual Servers and Objects

It is well known that servers store their binding information in CDS where clients can find it. But so far, CDS has been 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 accessible 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 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.

Part 4 provides more information about CDS and the structure and uses of CDS names. For 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 getting to the actual management tasks, let's examine a server entry to see exactly what is to be managed. See Figure 10 on page 124 for the type of information a server entry might contain.

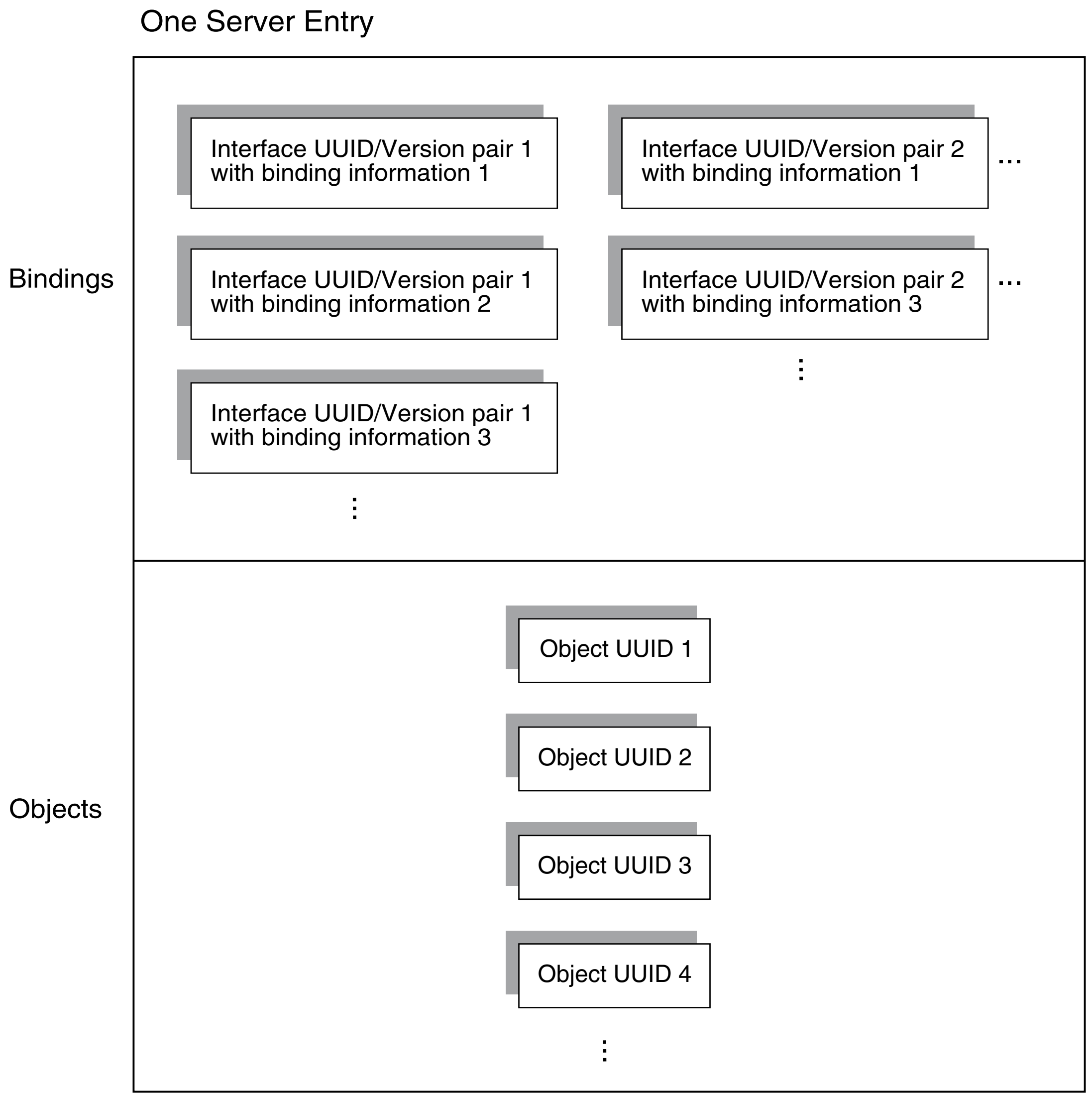

One Server Entry
Bindings
Interface UUID/Version pair 1 with binding information 1
Interface UUID/Version pair 2 with binding information 1
Interface UUID/Version pair 1 with binding information 2
Interface UUID/Version pair 2 with binding information 3
Interface UUID/Version pair 1 with binding information 3
Objects
Object UUID 1
Object UUID 2
Object UUID 3
Object UUID 4

Figure 10. 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. 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}
```

When an interface identifier is available over a 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 doesn't 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).

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>
```

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.

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>
```

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.

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>
```

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.

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>
```

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. 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>
```

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.

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>
```

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'd 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 doesn't 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. In Figure 11, it shows how a group entry contains members that point to other groups and to server entries.

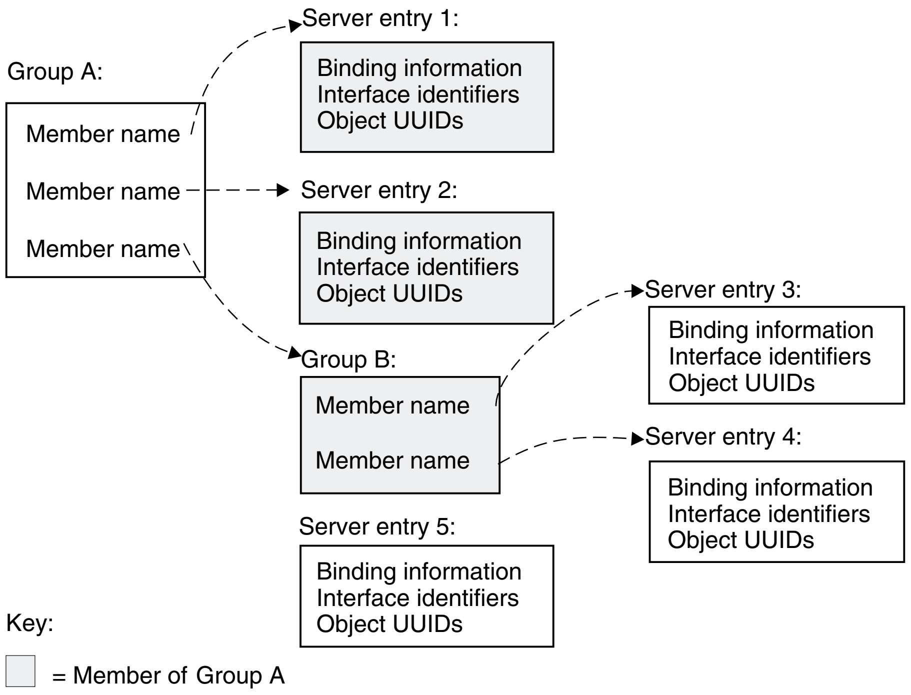


Figure 11. Possible Mappings of a Group

Now, let's 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.

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>
```

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.

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>
```

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.

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>
```

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 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.

```
dcebcp> rpcgroup import /.:/subsys/applications/admin_bbs_servers \
>-interface {458ffcbe-98c1-11cd-88bc-0000c08adf56 1.0}
{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.

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.

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>
```

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. 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>
```

Profiles Help Direct Clients' 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 set priorities for servers to be used by particular clients.

Profiles offer a complimentary way to organize servers because you can set priorities for 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 through 7; 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.

  Note that when you create a profile, you should use priorities 1 through 7 to manage profile elements. Priority 0 is a reserved priority that should only be used by the default profile element. If priority 0 is used by profile elements other than the default profile element, then DCE will choose a random binding from among the two. DCE does not always distinguish between priority 0 default elements and priority 0 non-default elements. Therefore, if you create a profile it is better to use priority 1 to denote the highest priority element in your profile instead of using priority 0. The other lower priority elements in your profile can be given priorities 2 through 7 leaving priority 0 reserved for the default element. This convention, along with the fact that only one default element is allowed per profile, will prevent the high priority given to default elements from being abused.

- 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. In Figure 12 on page 132, some possible mappings of a profile are shown.

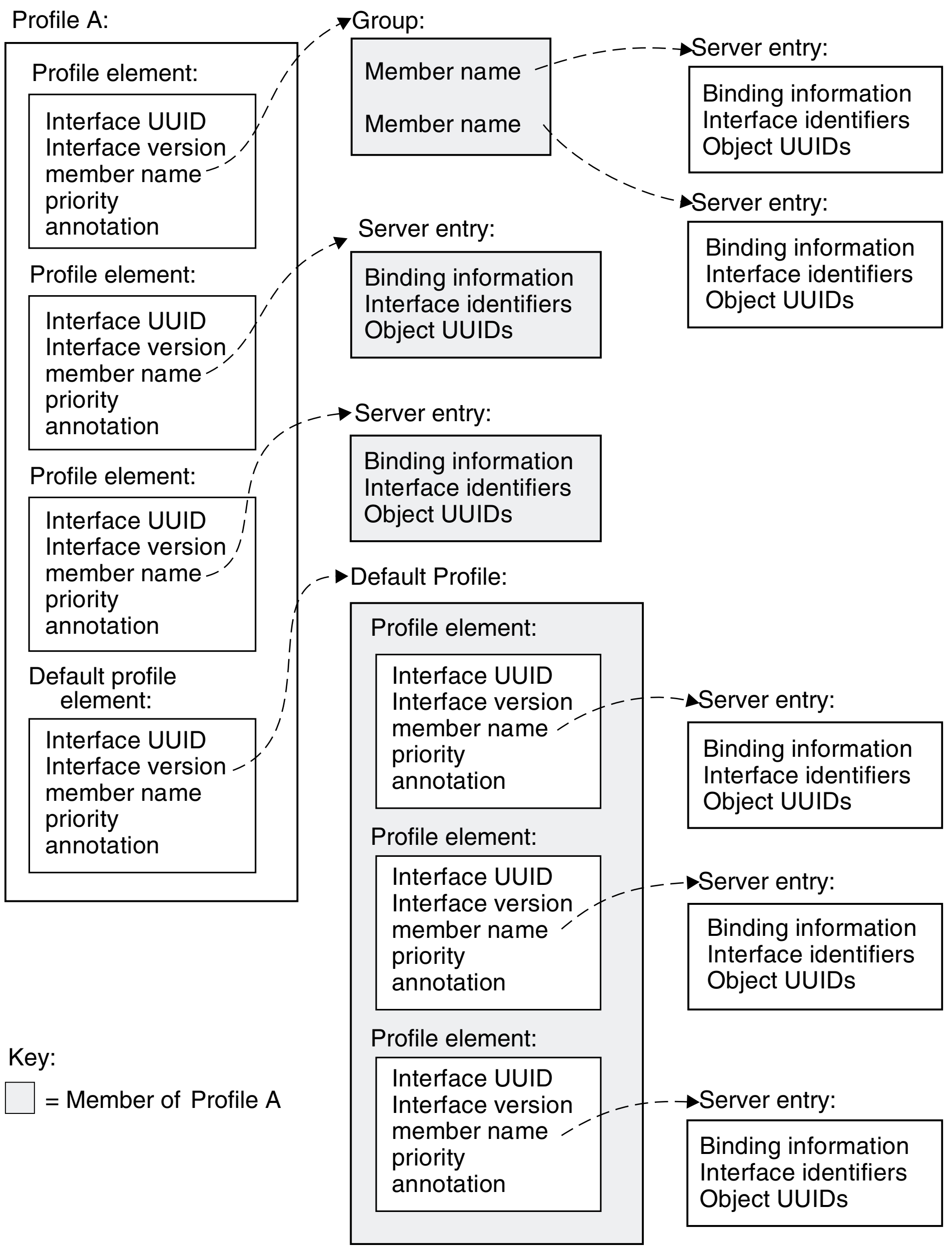


Figure 12. Possible Mappings of a Profile

To get an idea of how profiles can work, let's build on our printer example from the preceding discussion on group entries. The following profile entry example shows one way to use profiles to determine priorities for 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's 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 server 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.

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>
```

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.

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>
```

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.

To list the members of a profile entry in CDS, use an **rpcprofile list** 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>
```

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. 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>
```

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.

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>
```

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. 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>
```

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. 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>
```

Client Administration

So far, this chapter has focused on server administration issues. We've seen how to control some server operations, and how to store server binding information in CDS and in the host endpoint map where clients can find it. 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.

CDS is known to be a hierarchical system of directories that stores server binding information in the form of server entries. It is also known 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?

As discussed earlier in this chapter, 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 act 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, this name is being called just 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 earlier in this chapter, 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. 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
```

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. 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 on most systems). The following example shows a shell script that defines this variable and then calls the client.

```
#! /bin/sh
# Shell Script to start the InfoClient application
export RPC_DEFAULT_ENTRY=/.:/finance/operations/InfoServ_profile
infoclient
```

Chapter 15. Examples of Setting Up RPC Profiles

This chapter shows you two examples of setting up search paths to servers. Each example describes a user scenario that requires the use of RPC server entries, groups or profiles. The steps in arriving at the desired solution are described in each example.

The first example sets up an RPC profile to organize the search path to three mathematical servers. The second example is a more complex scenario involving print servers. In this example, setting up the appropriate search path involves the use of RPC server entries, groups, and profiles.

Note: In the examples, the back slash at the end of a command line (\) allows lengthy commands to be continued to the next line.

Math Server Example

In this example, three servers, MathA, MathB, and MathC that offer mathematical functions are made available to all clients. Figure 13 shows the functions offered by these servers.

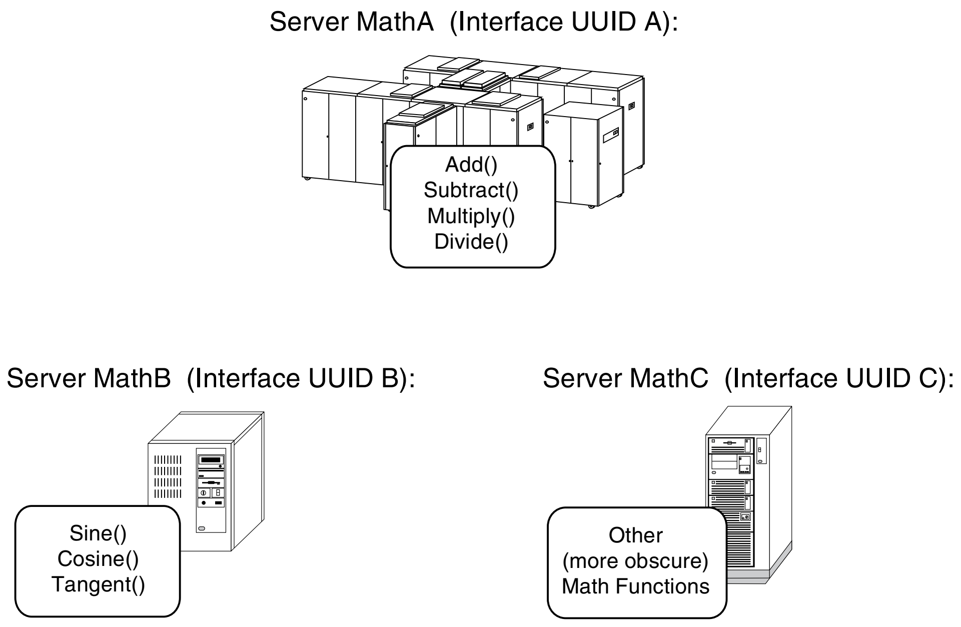

Server MathA (Interface UUID A):
Add()
Subtract()
Multiply()
Divide()
Server MathB (Interface UUID B):
Sine()
Cosine()
Tangent()
Server MathC (Interface UUID C):
Other
(more obscure)
Math Functions

Figure 13. Example: Math Servers

Frequency of Use

The frequency of use for these servers is as follows:

- Functions offered by MathA are the most often used.
- Functions offered by MathB are the second most often used.
- Functions offered by MathC are rarely used.

Also, clients always use the RPC_DEFAULT_ENTRY environment variable in setting the starting point when searching for compatible bindings.

Solution

You must create an RPC profile to arrange the order by which the servers are searched whenever a client makes a remote procedure call.

You must also specify this profile as the value for the RPC_DEFAULT_ENTRY environment variable to make it the starting point for finding a compatible server.

Construct the RPC profile such that the search order will follow this sequence:

1. Check the server entry for server MathA.
2. If server MathA is not compatible, check server MathB.
3. If server MathB is not compatible, check server MathC.

The RPC profile requires three elements, each referring to one of the server entries. The profile elements must be prioritized to follow the search order described above. The lowest priority element can be a default element that refers to a default profile.

Steps in Creating the RPC Profile

To create the RPC profile that defines the search order defined above, follow these steps:

1. Create a directory in the CDS namespace that will hold the server entries. Use the **directory create** command of **dcecp** to create this directory. In this example, the CDS directory is named **/.:/servdir**. For example, to create the directory **servdir**:

   ```
   dcecp> directory create /.:/servdir
   ```

Adding Server and Profile Entries to the Namespace

2. Add the server entries in the Directory Service namespace for MathA, MathB, and MathC using the **add entry** subcommand of RPCCP:

   ```
   rpccp> add entry /.:/servdir/MathA
   rpccp> add entry /.:/servdir/MathB
   rpccp> add entry /.:/servdir/MathC
   ```

3. Add an entry for the profile in the Directory Service namespace. In this example, this profile is named **MathProf**:

   ```
   rpccp> add entry /.:/servdir/MathProf
   ```

4. Add an entry for the default profile in the Directory Service namespace. In this example, this default profile is named **MathProfDef**:

   ```
   rpccp> add entry /.:/servdir/MathProfDef
   ```

Exporting Server Binding Information

5. Export the binding information for the server entries MathA, MathB, and MathC.

   ```
   rpccp> export /.:/servdir/MathA \
   >      -i EC1EEB60-5943-11C9-A309-08002B102989,1.0 \
   >      -b ncadg_ip_udp:128.20.15.20

   rpccp> export /.:/servdir/MathB \
   >      -i EC1EEB60-5943-11C9-A309-08057C304441,1.0 \
   >      -b ncadg_ip_udp:128.20.15.21

   rpccp> export /.:/servdir/MathC \
   >      -i EC1EEB60-5943-11C9-A309-092300065928,1.0 \
   >      -b ncadg_ip_udp:128.20.15.22
   ```

Adding Profile Elements

6. Add the profile element for MathA to the MathProf profile. Because you want this to be the first one to be searched, set its priority to 1.

```
rpccp> add element /.:/servdir/MathProf \
>      -m /.:/servdir/MathA \
>      -i EC1EEB60-5943-11C9-A309-08002B102989,1.0 \
>      -a arithmetic \
>      -p 1
```

7. Add the profile element for MathB to the MathProf profile. Because you want this to be searched after MathA, set its priority to 2.

```
rpccp> add element /.:/servdir/MathProf \
>      -m /.:/servdir/MathB \
>      -i EC1EEB60-5943-11C9-A309-08057C304441,1.0 \
>      -a trigonometry \
>      -p 2
```

8. Add the default element to refer to the default profile:

```
rpccp> add element /.:/servdir/MathProf \
>      -m /.:servdir/MathProfDef -d -a default
```

9. Add the profile element for MathC to the default profile.

```
rpccp> add element /.:/servdir/MathProfDef \
>      -m /.:/servdir/MathC \
>      -i EC1EEB60-5943-11C9-A309-092300065928,1.0 \
>      -a obscure_math \
>      -p 0
```

Setting the RPC_DEFAULT_ENTRY Environment Variable

10. Set the environment variable RPC_DEFAULT_ENTRY to the RPC profile. In OS/390 DCE, you can set this by editing the **$HOME/envar** file. For example:

```
RPC_DEFAULT_ENTRY=/.:/servdir/MathProf
```

Figure 14 shows the search path that is set up by the profile MathProf.

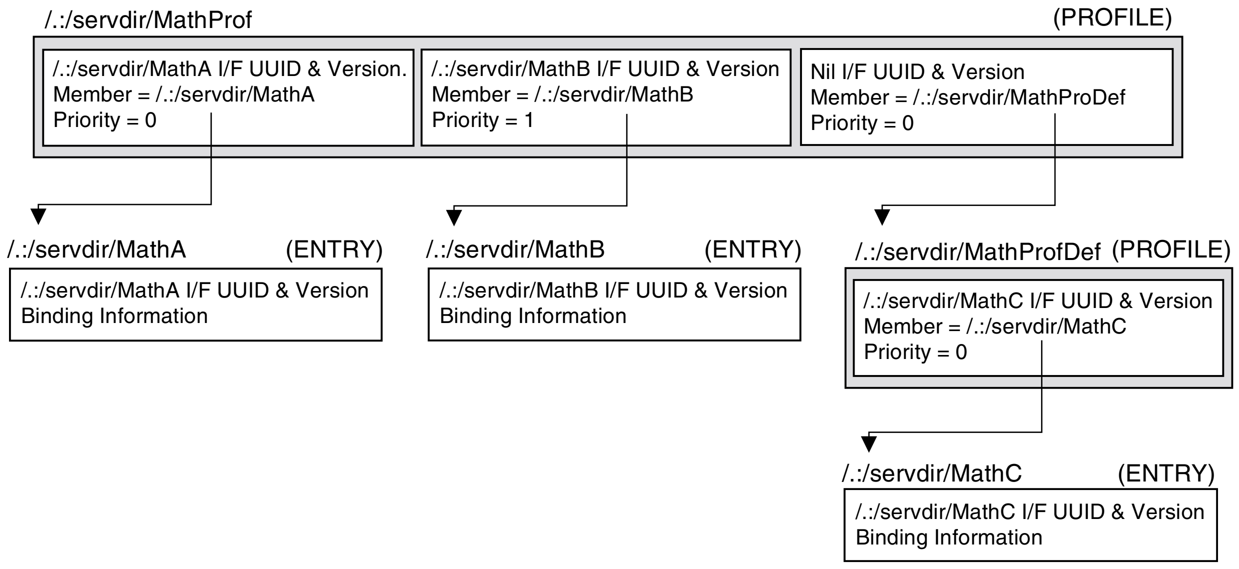

/.:/servdir/MathProf
(PROFILE)
/.:/servdir/MathA I/F UUID & Version.
Member = /.:/servdir/MathA
Priority = 0
/.:/servdir/MathB I/F UUID & Version
Member = /.:/servdir/MathB
Priority = 1
Nil I/F UUID & Version
Member = /.:/servdir/MathProDef
Priority = 0
/.:/servdir/MathA
(ENTRY)
/.:/servdir/MathA I/F UUID & Version
Binding Information
/.:/servdir/MathB
(ENTRY)
/.:/servdir/MathB I/F UUID & Version
Binding Information
/.:/servdir/MathProfDef
(PROFILE)
/.:/servdir/MathC I/F UUID & Version
Member = /.:/servdir/MathC
Priority = 0
/.:/servdir/MathC
(ENTRY)
/.:/servdir/MathC I/F UUID & Version
Binding Information

Figure 14. MathProf Search Path

Print Server Example

In this example, five print servers whose names are PrintA, PrintB, PrintC, PrintD, and PrintE are available for use in an office. The characteristics of each print server are as follows:

- PrintA controls a 3812 printer in an open area on the first floor.
- PrintB controls two printers: a 3820 and a 3800 in a restricted room on the first floor.
- PrintC also controls two printers: a 3820 and a 3800 in a restricted room on the first floor.
- PrintD controls two printers: a 3800 and a 38PP in a restricted room on the second floor.
- PrintE controls two printers: a 3812 and a 3820 in an open area on the second floor.

Figure 15 depicts this scenario.

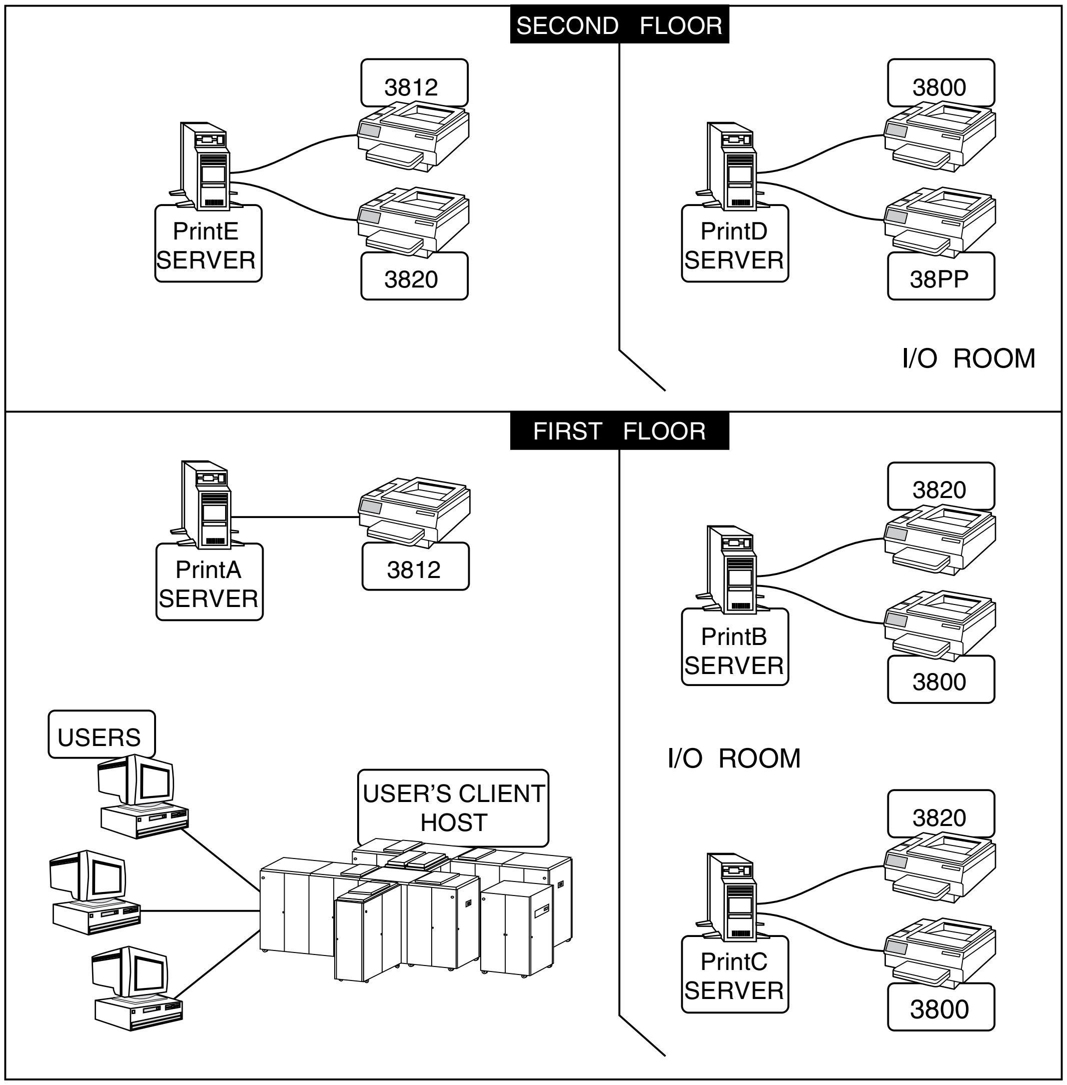


Figure 15. Print Servers

The following conditions exist in this company:

- All users are on the first floor.
- All users are connected to a specific host system.
- All users use the same *print* application to carry out their requests.

Frequency of Use

The frequency of use for these servers can be described as follows:

- In general, the preference for these types of printers are in the following order (from most preferred to least preferred):
  1. 3812
  2. 3820
  3. 3800
  4. 38PP
- Users prefer to use the nearest printer of the printer type that is requested.
- If possible, users prefer not to use the printers in the I/O rooms. If they have to use the printers in these rooms, they prefer the printers in the I/O room on the first floor.

Solution

You must create an RPC profile to arrange the search order whenever a client runs the *print* program. You can specify this profile as the value for the RPC_DEFAULT_ENTRY environment variable to use it as the starting point for finding a compatible print server.

Each element in the RPC profile will refer to a particular printer type. The priority of the elements that refer to the same printer type (and therefore the same interface) will be in accordance with the users' preference for these printers.

Print servers that have the same priority in the order of selection will be formed into RPC groups.

Steps in Creating the RPC Profile

To create the RPC profile for the print servers, follow these steps:

1. Create a directory that will hold the print server entries. For example, to create the **printdir** directory:

```
dcecp> directory create /.:/printdir
```

Adding Server Entries to the Namespace

2. Run RPCCP. Add the server entries to the Directory Service namespace for each combination of server and printer interface:

```
rpccp> add entry /.:/printdir/PrintA_3812
rpccp> add entry /.:/printdir/PrintB_3820
rpccp> add entry /.:/printdir/PrintB_3800
rpccp> add entry /.:/printdir/PrintC_3820
rpccp> add entry /.:/printdir/PrintC_3800
rpccp> add entry /.:/printdir/PrintD_3800
rpccp> add entry /.:/printdir/PrintD_38PP
rpccp> add entry /.:/printdir/PrintE_3812
rpccp> add entry /.:/printdir/PrintE_3820
```

Exporting the Interfaces

3. For PrintA, export the 3812 printer interface:

```
rpccp> export /.:/printdir/PrintA_3812 \
>      -i EC1EEB60-5943-11C9-A309-080264390822,1.0 \
>      -b ncadg_ip_udp:81.20.15.12
```

4. For PrintB, export the 3820 and the 3800 printer interfaces:

```
rpccp> export /.:/printdir/PrintB_3820 \
>      -i EC1EEB60-5943-11C9-A309-080000456000,1.0 \
>      -b ncadg_ip_udp:81.20.15.10

rpccp> export /.:/printdir/PrintB_3800 \
>      -i EC1EEB60-5943-11C9-A309-075603221982,1.0 \
>      -b ncadg_ip_udp:81.20.15.10
```

5. For PrintC, export the 3820 and the 3800 printer interfaces:

```
rpccp> export /.:/printdir/PrintC_3820 \
>      -i EC1EEB60-5943-11C9-A309-080000456000,1.0 \
>      -b ncadg_ip_udp:81.20.15.14

rpccp> export /.:/printdir/PrintC_3800 \
>      -i EC1EEB60-5943-11C9-A309-075603221982,1.0 \
>      -b ncadg_ip_udp:81.20.15.14
```

6. For PrintD, export the 3800 and the 38PP printer interfaces:

```
rpccp> export /.:/printdir/PrintD_3800 \
>      -i EC1EEB60-5943-11C9-A309-075603221982,1.0 \
>      -b ncadg_ip_udp:81.20.15.92

rpccp> export /.:/printdir/PrintD_38PP \
>      -i EC1EEB60-5943-11C9-A309-002000438761,1.0 \
>      -b ncadg_ip_udp:81.20.15.92
```

7. For PrintE, export the 3812 and the 3820 printer interfaces:

```
rpccp> export /.:/printdir/PrintE_3812 \
>      -i EC1EEB60-5943-11C9-A309-080264390822,1.0 \
>      -b ncadg_ip_udp:81.20.15.99

rpccp> export /.:/printdir/PrintE_3820 \
>      -i EC1EEB60-5943-11C9-A309-080000456000,1.0 \
>      -b ncadg_ip_udp:81.20.15.99
```

Creating RPC Groups of Interchangeable Servers

8. Form RPC groups for **interchangeable server instances**. PrintB_3820 and PrintC_3820, which offer the 3820 interface, can form a group. PrintB_3800 and PrintC_3800, which offer the 3800 interface, can form another group. First, add an entry in the Directory Service namespace for the 3820 printer group:

   ```
   rpccp> add entry /.:/printdir/PrintGrp_3820
   ```

   Then add an entry for the 3800 printer group:

   ```
   rpccp> add entry /.:/printdir/PrintGrp_3800
   ```

Adding Members to the RPC Groups

9. Add PrintB_3820 and PrintC_3820 as members of the PrintGrp_3820 group:

   ```
   rpccp> add member /.:/printdir/PrintGrp_3820 -m PrintB_3820
   rpccp> add member /.:/printdir/PrintGrp_3820 -m PrintC_3820
   ```

10. Add PrintB_3800 and PrintC_3800 as members of the PrintGrp_3800 group:

    ```
    rpccp> add member /.:/printdir/PrintGrp_3800 -m PrintB_3800
    rpccp> add member /.:/printdir/PrintGrp_3800 -m PrintC_3800
    ```

Adding Namespace Entries for the RPC Profiles

11. Add an entry to the Directory Service namespace for the RPC profile. In this example, the name of the profile is **PrintProf**.

```
rpccp> add entry /.:/printdir/PrintProf
```

```
rpccp> add entry /.:/printdir/PrintProfdef
```

Adding Elements to the RPC Profiles

12. Add the elements for the 3812 printers to the RPC Profile. They have the highest priority among all printers. Two servers access the 3812 printers: PrintA_3812 and PrintE_3812. PrintA_3812 is preferred over PrintE_3812. Add two elements to the profile:

```
rpccp> add element /.:/printdir/PrintProf \
>      -m /.:/printdir/PrintA_3812 \
>      -i EC1EEB60-5943-11C9-A309-080264390822,1.0 \
>      -a 3812_floor_1 \
>      -p 0

rpccp> add element /.:/printdir/PrintProf \
>      -m /.:/printdir/PrintE_3812 \
>      -i EC1EEB60-5943-11C9-A309-080264390822,1.0 \
>      -a 3812_floor_2 \
>      -p 1
```

13. Add the elements for the 3820 printers to the RPC Profile. The 3820 printers are second preference. Three print servers access 3820 printers: PrintB_3820, PrintC_3820, and PrintE_3820. PrintB_3820 and PrintC_3820 are less preferred than PrintE_3820. PrintB_3820 and PrintC_3820 are in the group PrintGrp_3820. Add two elements to the profile:

```
rpccp> add element /.:/printdir/PrintProf \
>      -m /.:/printdir/PrintE_3820 \
>      -i EC1EEB60-5943-11C9-A309-080000456000,1.0 \
>      -a 3820_floor_2 \
>      -p 2

rpccp> add element /.:/printdir/PrintProf \
>      -m /.:/printdir/PrintGrp_3820 \
>      -i EC1EEB60-5943-11C9-A309-080000456000,1.0 \
>      -a 3820_floor_1_IO \
>      -p 3
```

14. Add a default element in the RPC profile to refer to the default profile. The objective here is to place the servers for the 3800 and the 38PP printers in the default profile for general use.

```
rpccp> add element /.:/printdir/PrintProf \
>      -m /.:/printdir/PrintProfDef -d -a default
```

15. Add elements to the default profile. Elements for the 3800 printers must be added to the default profile. Three print servers access the 3800 printers: PrintB_3800, PrintC_3800, and PrintD_3800. PrintB_3800 and PrintC_3800 are preferred over PrintD_3800. PrintB_3800 and PrintC_3800 have also been made members of the group PrintGrp_3800. Add two elements to the default profile:

```
rpccp> add element /.:/printdir/PrintProfDef \
>      -m /.:/printdir/PrintGrp_3800 \
>      -i EC1EEB60-5943-11C9-A309-075603221982,1.0 \
>      -a 3800_floor_1_IO
>      -p 0

rpccp> add element /.:/printdir/PrintProfDef \
>      -m /.:/printdir/PrintD_3800 \
>      -i EC1EEB60-5943-11C9-A309-075603221982,1.0 \
>      -a 3800_floor_2_IO
>      -p 1
```

16. Add an element in the default profile for the 38PP print server:

```
rpccp> add element /.:/printdir/PrintProfDef \
>      -m /.:/printdir/PrintD_38PP \
>      -i EC1EEB60-5943-11C9-A309-002000438761,1.0 \
>      -a 38PP_floor_2_IO
>      -p 2
```

17. Set the RPC_DEFAULT_ENTRY environment variable to the PrintProf RPC profile.

Figure 16 on page 147 displays the search path of this RPC profile.

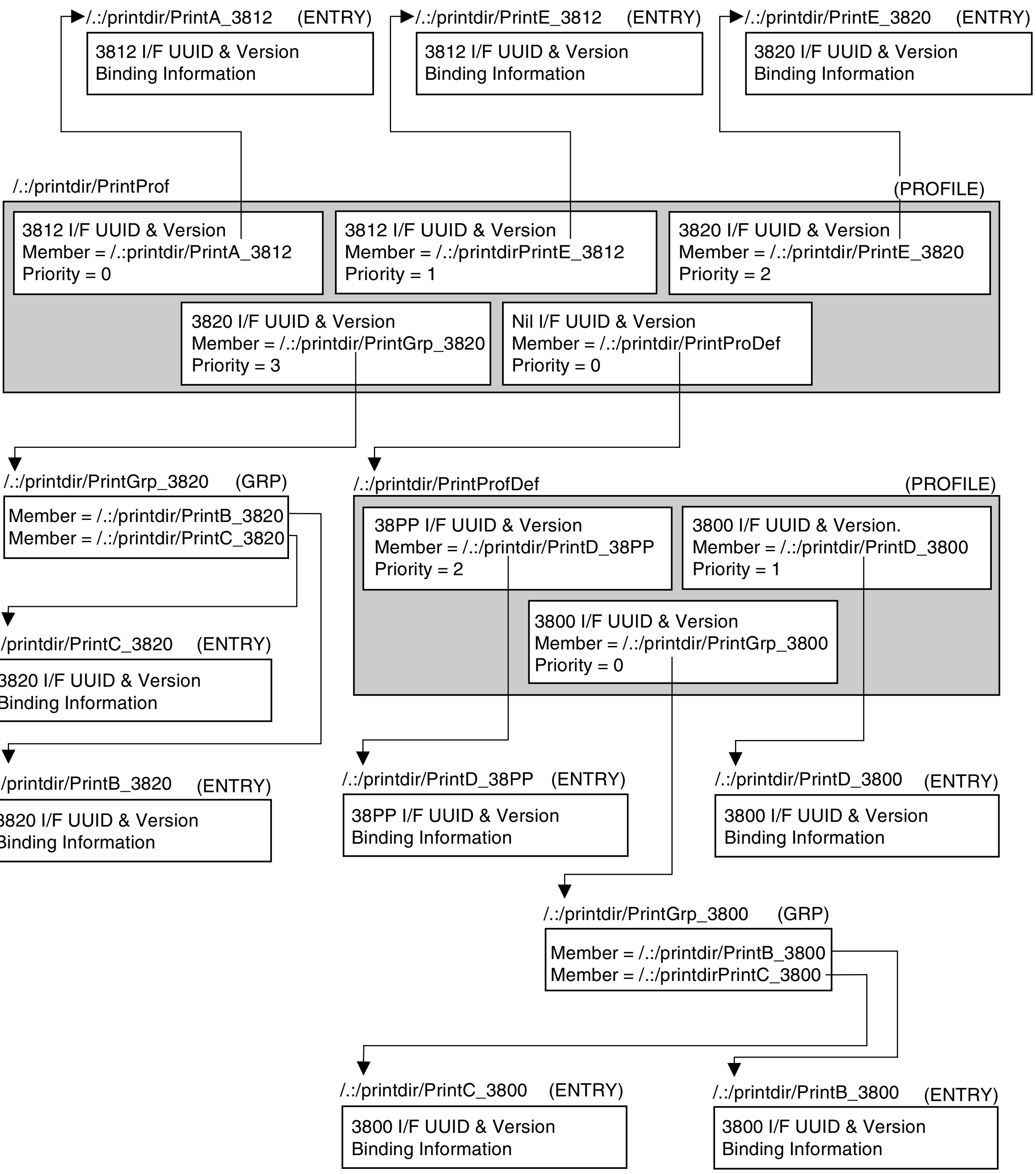

/.:/printdir/PrintA_3812 (ENTRY)
3812 I/F UUID & Version
Binding Information
/.:/printdir/PrintE_3812 (ENTRY)
3812 I/F UUID & Version
Binding Information
/.:/printdir/PrintE_3820 (ENTRY)
3820 I/F UUID & Version
Binding Information
/.:/printdir/PrintProf
(PROFILE)
3812 I/F UUID & Version
Member = /.:printdir/PrintA_3812
Priority = 0
3812 I/F UUID & Version
Member = /.:/printdirPrintE_3812
Priority = 1
3820 I/F UUID & Version
Member = /.:/printdir/PrintE_3820
Priority = 2
3820 I/F UUID & Version
Member = /.:/printdir/PrintGrp_3820
Priority = 3
Nil I/F UUID & Version
Member = /.:/printdir/PrintProDef
Priority = 0
/.:/printdir/PrintGrp_3820 (GRP)
Member = /.:/printdir/PrintB_3820
Member = /.:/printdir/PrintC_3820
/.:/printdir/PrintProfDef
(PROFILE)
38PP I/F UUID & Version
Member = /.:/printdir/PrintD_38PP
Priority = 2
3800 I/F UUID & Version.
Member = /.:/printdir/PrintD_3800
Priority = 1
3800 I/F UUID & Version
Member = /.:/printdir/PrintGrp_3800
Priority = 0
/.:/printdir/PrintC_3820 (ENTRY)
3820 I/F UUID & Version
Binding Information
/.:/printdir/PrintB_3820 (ENTRY)
3820 I/F UUID & Version
Binding Information
/.:/printdir/PrintD_38PP (ENTRY)
38PP I/F UUID & Version
Binding Information
/.:/printdir/PrintD_3800 (ENTRY)
3800 I/F UUID & Version
Binding Information
/.:/printdir/PrintGrp_3800 (GRP)
Member = /.:/printdir/PrintB_3800
Member = /.:/printdirPrintC_3800
/.:/printdir/PrintC_3800 (ENTRY)
3800 I/F UUID & Version
Binding Information
/.:/printdir/PrintB_3800 (ENTRY)
3800 I/F UUID & Version
Binding Information

Figure 16. Search Path for Print Servers

Chapter 16. Controlling Access to the DCED Endpoint Map

Application servers **register** their dynamic endpoints with DCED, which stores the endpoints in the endpoint map. The endpoint map is a critical database that must be protected from unauthorized access. If the integrity of the endpoint map is compromised, DCE clients may be unable to access any application servers.

This chapter describes how you can restrict access to the DCED endpoint map and how you can protect the endpoint map. The first part of the chapter provides an overview of DCE authorization to the DCED endpoint map. This includes a description of the entry types and permissions that are supported in the Access Control List (ACL) for the DCED endpoint map. The second part of the chapter describes the steps that you must perform to control access to the DCED endpoint map.

You should be familiar with the ACL facility provided by the DCE Security Service before reading this section. ACLs are described in detail in the Security part of this book.

Overview of DCED Endpoint Map Authorization

There are three types of users of DCED:

- DCE clients who must obtain the dynamic endpoints of the application servers
- Application servers that register or unregister the interfaces that they support
- DCE administrators who register or unregister interfaces, rebuild the endpoint map, or query the status of DCED.

Access to DCED must be controlled so that these users are given only the appropriate permissions required to perform their tasks, while preventing unauthorized principals from accessing DCED. In OS/390 DCE, an ACL Manager for DCED controls access to the DCED endpoint map.

An ACL Manager is the part of a server that determines a principal's authorization to perform an operation on the server and the objects that it controls. The ACL Manager reads an ACL that defines the DCE users (human users, servers, or machines) who can access an object and the operations they are allowed to perform on it. The operations that a user can perform on an object are called the user's **permissions** to that object.

You can control access to DCED by changing the Access Control List associated with the DCED object. You can perform this by editing the ACL of the DCED endpoint map object using the ACL Editor facility, or by using the **dcecp acl** object.

You can also define groups of users (called Security groups) to enforce a specific set of permissions to an object on several principals. The members of a Security group are principals who will share the same permissions to a certain object. That is, each of the users in the group will inherit the permissions that have been given to that group.

ACL Entry Types Supported by the DCE Endpoint Map ACL Manager

See Table 15 on page 311, which lists, explains, and displays the format of the ACL entry types supported by the DCED Endpoint Map ACL Manager.

Permissions Used for the DCE Daemon Endpoint Map

Table 5 on page 150 lists the permissions used by the DCE daemon.

Table 5. Permissions Used by the DCE Daemon

Permission	Meaning
c	Control access. Allows a principal to change the ACL database, as well as all access rights associated with the **l**, **i**, **d**, **x**, and **t** permissions. These permissions are also described below.
l	List access. Allows a principal to list the ACL entries.
i	Allows a principal to register endpoints in the DCED endpoint map.
d	Allows a principal to unregister endpoints in the DCED endpoint map.
s	Server permission. Allows a principal to register and unregister endpoints in the DCED endpoint map. Equivalent to the **i** and **d** permissions.
x	Execute. Allows a principal to rebuild the endpoint map. Not currently available in OS/390 DCE.
t	Allows a principal to test the access rights to the object.

DCE Clients' Access to DCED

DCE clients only need read access to the DCED endpoint map to obtain fully-bound binding handles to the application servers. By default, DCE clients have read access to DCED. No administrative tasks are involved in enabling DCE clients to obtain dynamic endpoints to the application servers.

Giving Application Servers Access to the DCED Endpoint Map

Application servers must have the appropriate permissions to the DCED object to **register** and **unregister** their supported interfaces. In OS/390 DCE, access control to DCED by application servers is made easier by supplying a default defined Security group whose members can consist of application servers. This Security group will be referred to as the **RPC server group** in this book.

RPC Server Group

A server group is created during the DCE configuration of the host that has all the permissions required to register and unregister interfaces to DCED. When you run the DCE configuration program, **DCECONF**, this server group is created with the name **subsys/dce/rpc-server-group**. The DCE configuration program then provides the required permissions to this group to access DCED. The DCE configuration of the host is discussed in detail in the *OS/390 DCE: Configuring and Getting Started*.

The RPC server group provides a simple way by which application servers can gain the proper access privileges to DCED. You can add an application server as a member of the server group, for it to inherit the group's access privileges to DCED, including the privilege to register and unregister interfaces. You add the application server as a member of the server group by editing the Security registry database using the Registry Editor.

Note: To make the application server a member of the server group, a DCE account must first be created for the application server, using the Registry Editor. Creating a DCE account for a server is discussed in the Security part of this book.

The following summarizes the steps you take to give an application server the required permissions to access DCED.

1. Make sure that a DCE account has been created for the application server.
2. Run the Registry Editor and change to the group domain:

```
rgy_edit=> domain group
Domain changed to: group
```

3. Add the application server as a member of the RPC Server Group. For example, to make an application server whose principal name is **app** a member of the server group **hosts/dce-host/rpc-server-group**:

   ```
   rgy_edit=> member
   Enter group name: hosts/dce-host/rpc-server-group
   Enter name to add: app
   ```

Note: You do not have to use only the RPC server group that was defined during the initial DCE configuration of the host. You can also create your own server group and then add your application servers to this group, using the Registry Editor. If you elect to create your own server group, add an entry in the ACL of DCED that will give this group the appropriate privileges to DCED, as described in "Permissions Used for the DCE Daemon Endpoint Map" on page 149.

Giving DCE Administrators Access to the DCED Endpoint Map

Administrators need access to DCED to perform the following tasks:

- Register the interfaces of application servers
- Delete or **unregister** interfaces of application servers
- Rebuild the endpoint map in case of a system malfunction

Administrators are given access to DCED by changing the ACL that is associated with the DCED endpoint map object. The administrator must be given the **c** (control) permission to the DCED object. This permission includes the **l**, **i**, **d**, **s**, and **t** permissions in addition to the privilege to change the ACL database. These permissions are explained in "Permissions Used for the DCE Daemon Endpoint Map" on page 149.

Note: The administrator principal used to perform the DCE configuration of the OS/390 host is automatically given the **c** permission to DCED. The steps described in this section are for administrators other than the one who performed the DCE configuration of the host.

To give the administrator the necessary permissions to the DCED endpoint map, you must:

1. Run the ACL Editor on the DCED object:

   ```
   ACLEDIT /.:/hosts/hostname/config/epmap (in TSO)
   ```

   or

   ```
   acl_edit /.:/hosts/hostname/config/epmap (in the shell)
   ```

   where *hostname* is the DCE host name of the OS/390 host system.

   Note: The ACL for the DCED object, **/.:/hosts/***hostname***/config/epmap**, is created during the initial configuration of the OS/390 host by the OS/390 DCE configuration program.

2. Change the ACL of DCED. For example, if the administrator's principal name is **admin1**:

   ```
   sec_acl_edit> modify user:admin1:c
   ```

Alternatively, you can also create a Security group whose members consist of administrators. In this case, give the group the appropriate access privilege (**c** permission) to DCED. Adding an administrator principal as a member of this group (using the Registry Editor) gives that administrator the **c** permission to DCED.

The Default Endpoint Map ACL

DCECONF creates these initial entries in the **config/epmap** ACL, assuming that **host_name** was the principal used when configuring the DCE host:

```
$ acl_edit /.:/hosts/<hostname>/config/epmap
sec_acl_edit> li

# SEC_ACL for /.:/hosts/<hostname>/config/epmap:
# Default cell = /.../cellname.yournode.com
unauthenticated:-l----t
user:hosts/<hostname>/self:clidsxt
user:host_admin:clidsxt
group:subsys/dce/rpc-server-group:-l--s-t
any_other:-l----t
```

This ensures that the endpoint map is protected by a default. See *OS/390 DCE: Configuring and Getting Started* for a discussion of how to configure the ACL for no protection. Or, see “Giving Application Servers Access to the DCED Endpoint Map” on page 150 on how to add other servers.

Giving Unauthenticated Users Access to the DCED Endpoint Map

You can give unauthenticated users access to the DCE daemon by adding an **unauthenticated** entry in the access control list of the DCE daemon which gives the **s** permission to unauthenticated users to access DCED. For example, you can start the ACL Editor as follows:

ACLEDIT /.:/hosts/*hostname***/config/epmap**

where *hostname* is the name of the OS/390 host.

Then, modify the ACL of the DCED endpoint map as follows:

```
sec_acl_edit> modify unauthenticated:s
```

To allow completely unrestricted access to the endpoint map, give all entries **clidsxt** privileges and create the entry **any_other** with these privileges. Note that in this case all remote users have full access to insert or delete entries in the DCED endpoint map.

Chapter 17. Workload Balancing in a Parallel Sysplex Environment

Modifications to the endpoint map in DCE allow workload balancing among CPUs in a Parallel Sysplex environment.

Overview of Workload Balancing

Whenever a program contains a remote procedure call (RPC), the calling program is acting as a client and the remote procedure is in the role of a server. In an S/390® Parallel Sysplex environment, identical procedure code may reside on more than one server and even on different hosts. Depending on how the RPC is assigned a server, the resource utilization on the different hosts will not necessarily be balanced. Using the Parallel Sysplex hardware feature called the Coupling Facility (XCF) and the program called Workload Manager (WLM) allows workloads to be balanced among different hosts.

During an RPC, the client goes to the Cell Directory Server (CDS) to locate a server instance. CDS returns a list of server instances, and then the runtime dynamic load library (DLL) selects the one to use. Finding the host is the first step; finding the server is next.

On an S/390 processor the concept of "host" really means an **image**. An image can be a separate machine (separate CPU) or a logical partition (LPAR). In a Parallel Sysplex environment, with XCF and WLM, multiple images are able to share data. WLM distributes parts of the operating system across images and allows balancing of workloads within the whole Sysplex rather than only within an image. For more information on WLM, see :citdocid=ieaw100.OS/390: MVS Planning: Workload Management (GC28-1761) and *OS/390: MVS Planning: Workload Management Services* (GC28-1773).

To a client, a server instance provides services through the interfaces that it registers with its runtime DLL and with the endpoint mapper on its local host. Until now, the selection of the server instance was done independently of resource availability on the server host. Requests would be distributed to server instances equally even if one instance were on a heavily-loaded host while another was on a lightly-loaded one. With DCE workload balancing, a server registers an **aename**, which is a name shared by all server instances providing identical services, and it also registers which interfaces are available to be balanced across server instances. These are then registered with WLM and with the local endpoint map on behalf of the server instance.

Certain steps occur when:

- The endpoint map receives a request to resolve a binding, *or*
- The endpoint map forwards a request for an interface that has been registered as supporting load balancing.

These are the steps:

1. The endpoint mapper uses the aename that was registered and asks WLM for a list of servers.
2. WLM returns the list of servers together with a set of weights reflecting resource availability for the hosts where the server instances reside.
3. The endpoint mapper chooses a server instance from the list and either resolves the binding or forwards the request to that server instance.
4. The client continues to communicate with the same server instance until it acquires a new binding or until it resets the current one.

Setting Up Workload Balancing

Before workload balancing can be implemented in a Parallel Sysplex environment, the servers and clients must meet certain requirements.

Server Requirements

- The server must support multiple instances.
- Internal state data must be shared across all instances. For example, there must be a lock mechanism that works across images when a database is modified. In addition, if there is caching, there must be a cache-management mechanism that works across the Sysplex.
- Workload balancing must be occurring within a single Sysplex.
- If object UUIDs are used, either
  - All server instances must use the same object UUID, *or*
  - All server instances must accept all possible object UUIDs.
- The aename must be set before any of these APIs is used for the first time:
  - **rpc_ep_register**
  - **rpc_ep_register_wlb**
  - **rpc_ep_no_replace**
  - **rpc_ep_no_replace_wlb**
- The server host must be running OS/390 DCE V2R6 or higher.

Client Requirements

- If the client uses the TCP protocol or uses the **rpc_ep_resolve_binding** API, the level of OS/390 DCE on the client must be V2R6 or higher.
- If the client uses only User Datagram Protocol (UDP) call forwarding, the client can be running any level of OS/390 DCE.
- The client must resolve a new binding regularly.
- The client must resolve the binding using a load-balanced interface.

Using Workload Balancing

There are two different ways to use the workload-balancing support of OS/390 DCE. You can use the environment variable, _EUV_LOAD_BALANCE, or you can use the application programming interfaces (APIs). Whenever possible, it is preferable to use the environment variable because it requires fewer changes to be made to the server.

Using the Environment Variable

You can use the environment variable, _EUV_LOAD_BALANCE, to provide the value of *aename* to the runtime DLL for workload-balanced interfaces. This value is overridden by any value set using the **rpc_set_ae_name** API.

Using the Application Programming Interfaces

There are six APIs that are specifically for workload balancing. These are:

- **rpc_ep_register_wlb**
- **rpc_ep_register_no_replace_wlb**
- **rpc_set_ae_name**
- **rpc_get_ae_name**
- **rpc_mgmt_ep_elt_inq_next_wlb**
- **rpc_mgmt_ep_set_activated_wlb**

The APIs for workload balancing are described in *OS/390 DCE: Application Development Reference*. For information on how to use RPC APIs, see the *OS/390 DCE: Application Development Guide: Core Components*.

Using dcecp endpoint Commands with Workload Balancing

There are several **dcecp endpoint** commands that you may need while using the workload balancing support of OS/390 DCE. These commands are:

- **endpoint wlb_activate**
- **endpoint wlb_ae_show**
- **endpoint wlb_off**
- **endpoint wlb_on**
- **endpoint wlb_show**

See the *OS/390 DCE: Command Reference* for how to use these commands.

The PI Program

A sample program called PI is included on the OS/390 DCE product tape. You can use this program as a model for building your own applications. In addition, running the program allows you to see whether DCE is using WLM correctly. You may want to run PI at regular intervals for this purpose. You should occasionally delete the data log file that it produces.

Part 6. DCE Directory Service

Chapter 18. Introduction to the DCE Directory Service . . . 161
How the DCE Components Use Directory Services . . . 161
How to Use Directory Service . . . 162
Directory Services and the Cell Environment . 163
How Cells Determine Naming Environments . 165
Global Names . . . 165
Cell-Relative Naming in a Standalone Cell 166
Local Filenames . . . 167
A Closer Look at DCE Names . . . 167
Cell Directory Service Names . . . 167
Global Directory Service Names . . . 168
Domain Name System Names . . . 170
Names Outside of the DCE Directory Service . . . 171

Chapter 19. Cell Directory Service Concepts . . . 173
How the Cell Directory Service Works . . . 173
Replicas and Their Contents . . . 175
Object Entries . . . 176
Soft Links . . . 176
Child Pointers . . . 177
Putting It All Together . . . 177
CDS Advertiser . . . 177
CDS Advertiser and Clerk in OSF DCE . . . 178
CDS Advertiser and Clerk in OS/390 DCE . 179
Security in the Cell Directory Environment . . 179
Protecting the CDS Cache File . . . 181
Conversion between ASCII and EBCDIC in OS/390 DCE . . . 181
Cell Directory Service User Interface . . . 181

Chapter 20. How the Cell Directory Service Looks Up Names . . . 183
Translating from Names to Resources . . . 183
How the Cell Directory Service Finds Names 187
The Solicitation and Advertisement Protocol . . . 188
Lookups . . . 188
The cdscache create Command . . . 188

Chapter 21. How the Cell Directory Service Updates Data . . . 191
Update Propagation . . . 191
Skulk Operation . . . 191
How Timestamps Help Keep Data Consistent 192

Chapter 22. Managing the DCE Directory Service . . . 193
Using the dcecp Program . . . 193
CDS Managed Objects . . . 193
dcecp Command Operations for CDS . . . 193
CDS Object Attributes . . . 194
Using the cdscp Program . . . 194
Starting and Exiting . . . 194
The cdscp Program Commands . . . 195

Chapter 23. Controlling Access to CDS Names . . . 197
Overview of DCE Authorization for the Cell Directory Service . . . 197
ACL Types Supported by the Cell Directory Service . . . 197
How Permissions Propagate to CDS Directories and Their Contents . . . 198
Access Control List Entry Types . . . 198
DCE Permissions Supported by the Cell Directory Service . . . 200
Controlling Access to CDS Clerk and Server Management Operations . . . 200
Control Program Commands and Required Permissions . . . 201
Editing ACLs on Cell Directory Service Names . . . 203
How CDS Servers Gain Access to the Namespace . . . 204
Setting Up Access Control in a New Namespace . . . 204
Adding Members to the Namespace Authorization Group . . . 204
Creating Additional Authorization Groups . 205
Establishing Maximum Permissions for Unauthenticated Principals . . . 205

Chapter 24. Managing Clerks, Servers, and Clearinghouses . . . 207
Monitoring Clerk, Server, and Clearinghouse Counters . . . 207
Displaying Clerk Counters . . . 207
Displaying Server Counters . . . 207
Displaying Clearinghouse Counters . . . 207
Monitoring Clerk Communication with Specific Clearinghouses . . . 207
Displaying the Contents of a Clearinghouse . 208
Forcing the Clearinghouse to Checkpoint to Disk . . . 208
Disabling Clerks and Servers . . . 208

Disabling a Clerk 208
Disabling a Server 208
Stopping the CDS Advertiser and CDS Clerk 209
Starting and Stopping the CDS Daemon . . . 209
Restarting the CDS Advertiser and CDS Clerk 209
Automatic Restart after Abnormal Termination 210
Removing Stale Cache Entries 210
Recovering from a Corrupted CDS Cache . . 211
If Cache Size Is Changed 211
Preserving a Clearinghouse across a Server System Upgrade 211
Recovering from a cdsd File System Full Condition 212
Reconfiguring a Secondary CDS Server After Deconfiguring 216
Backing Up Namespace Information 217
Using Replication to Back Up Namespace Information 217
Using Operating System Backups 217

Chapter 25. Managing CDS Directories . . 219
Creating a Directory 219
Permissions for Creating a Directory . . . 219
Entering the directory create Command . . 219
Checking the ACL Entries for a New Directory 220
Upgrading the Directory Version on the Cell Root Directory 220
Upgrading the Directory Version on a Directory 220
Creating a Read-Only Replica 221
Before You Create a Replica 221
Permissions for Creating Replicas 223
Entering the directory create Command . . 223
Deleting a Read-Only Replica 223
Permissions for Deleting a Replica 223
Entering the directory delete Command . . 223
Skulking a Directory 224
Permissions for Skulking a Directory . . . 224
Entering the directory synchronize Command 224
Synchronizing CDS Server Clocks 225
Changing a Directory's Convergence 225
Directory Convergence Set to High 225
Directory Convergence Set to Medium . . 225
Directory Convergence Set to Low 226
Before You Modify a Directory's Convergence 226
Permissions for Modifying a Directory's Convergence 226
Entering the directory modify Command . 226

Chapter 26. Viewing the Structure and Contents of a Namespace 227
Listing the Contents of Directories 227
Displaying the Attribute Values of CDS Names 227
Displaying Clerk and Server Attribute Information 229
Preferred Clearinghouse for Viewing the Namespace 229

Chapter 27. Using the CDS Subtree Commands to Restructure CDS Directories 231
Overview of the Merge and Append Procedures 231
Merging CDS Directories 232
Appending CDS Directories 233
Modifying ACLs at the Target Location . . 234
Handling Errors 235
Duplicate Names 235
Unreachable Name Failures 235
Insufficient Permissions 235
Merging CDS Directories into a Foreign Cell .236
Establishing Cross-Cell Authentication . . 236
Performing a Merge Operation into a Foreign Cell 236
Restoring Merged CDS Directories 236

Chapter 28. Restructuring a Namespace . 239
Managing Soft Links 239
Creating a Soft Link 239
Setting Expiration and Extension Values for a Soft Link 239
Changing a Soft Link Destination Name . 240
Changing a Soft Link Expiration or Extension Value 240
Deleting a Soft Link 241
Changing a Directory Replica Set 241
Before You Change a Directory's Replica Set 242
Permissions Required for Modifying a Replica Set 242
Designating a New Master Replica 242
Excluding a Replica from a Replica Set . . 243
Deleting Directories 244
Deleting a Non-Replicated Directory 245
Deleting a Directory Replica 245
Relocating a Clearinghouse 246
Dissociating a Clearinghouse from Its Host Server System 247
Copying the Clearinghouse Database Files to the Target Server System 247
Starting the Clearinghouse on the Target Server 248

Deleting a Clearinghouse 249
  Before You Delete a Clearinghouse 249
  Permissions for Deleting a Clearinghouse .249
  To Delete a Clearinghouse 250

Chapter 29. Managing Intercell Naming . 251
How the Global Directory Agent Works . . . 251
Managing the Global Directory Agent 254
Enabling Other Cells to Find Your Cell 254
  Defining a Cell in the Domain Name Service 255
  Defining a Cell in an LDAP Server 256

Chapter 18. Introduction to the DCE Directory Service

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 telephone 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 division, such as European Sales; a location, such as the first floor of Building A; or a telephone number. Users can search for a name by supplying one or more of its attributes. For example, given the search value of **John Smith** and **Chicago**, the Directory Service could produce a list of telephone numbers for users in Chicago named John Smith. Search capabilities are currently limited to the global part of the DCE Directory Service environment.

This chapter first describes how the Directory Service is used both by clients and other DCE services. Then, it introduces the concept of a cell and explains how the Directory Service environment works with regard to cells. It introduces the main Directory Service components: the Cell Directory Service (CDS), the Global Directory Service (GDS), and the Global Directory Agent (GDA), which is a gateway between the local and global naming environments. The chapter also discusses DCE support for the Domain Name System (DNS), and for the Lightweight Directory Access Protocol (LDAP), existing name services that are not part of the DCE technology offering.

Note: GDS is currently not available in OS/390 DCE. This service is introduced here because it may be available on other hosts. For detailed discussions on how to administer GDS, refer to the documentation provided by the specific host that offers the service.

How the DCE Components Use Directory Services

The DCE Directory Service is a fundamental service that applications can rely on and use to their advantage.

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 run 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 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 find out 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.

How to Use Directory Service

Except for DCE administrators, directory services users usually use the Directory Service 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. The following examples, both real and hypothetical, illustrate some of the ways people can use the Directory Service:

- A user runs 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 use of the spell checker involved a call to the Directory Service and interaction with a remote server.
- A user logging in to 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, that 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.

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 of a cell are different. The main components of the DCE naming environment are:

- Cell Directory Service (CDS)
- Global Directory Service (GDS)
- Domain Name System (DNS)
- LDAP server
- Global Directory Agent (GDA).

CDS is a high-performance distributed service that provides a consistent, location-independent method for naming and using resources inside a cell (**intracell**).

GDS supports the global naming environment inside cells (**intracell**) and outside of cells (**intercell**). 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.

LDAP is a short name for the LDAP server, a server that supports the **Lightweight Directory Access Protocol (LDAP)**.

Figure 17 shows hypothetical configurations of cells that use GDS, DNS, or LDAP to access names in the other cells. CDS is the directory service within each cell. The same organization administers both cells, which are configured based on geographic location and network topology.

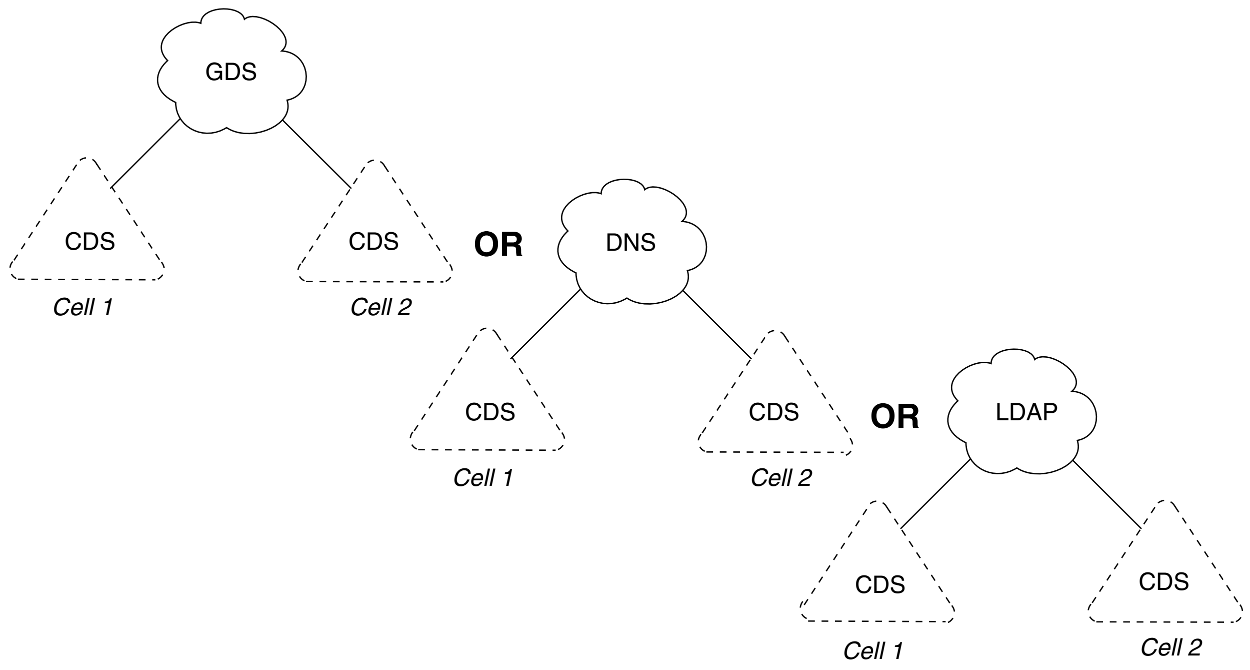


Figure 17. Cell and Global Naming Environments

DNS is a widely used existing global name service for which the 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.

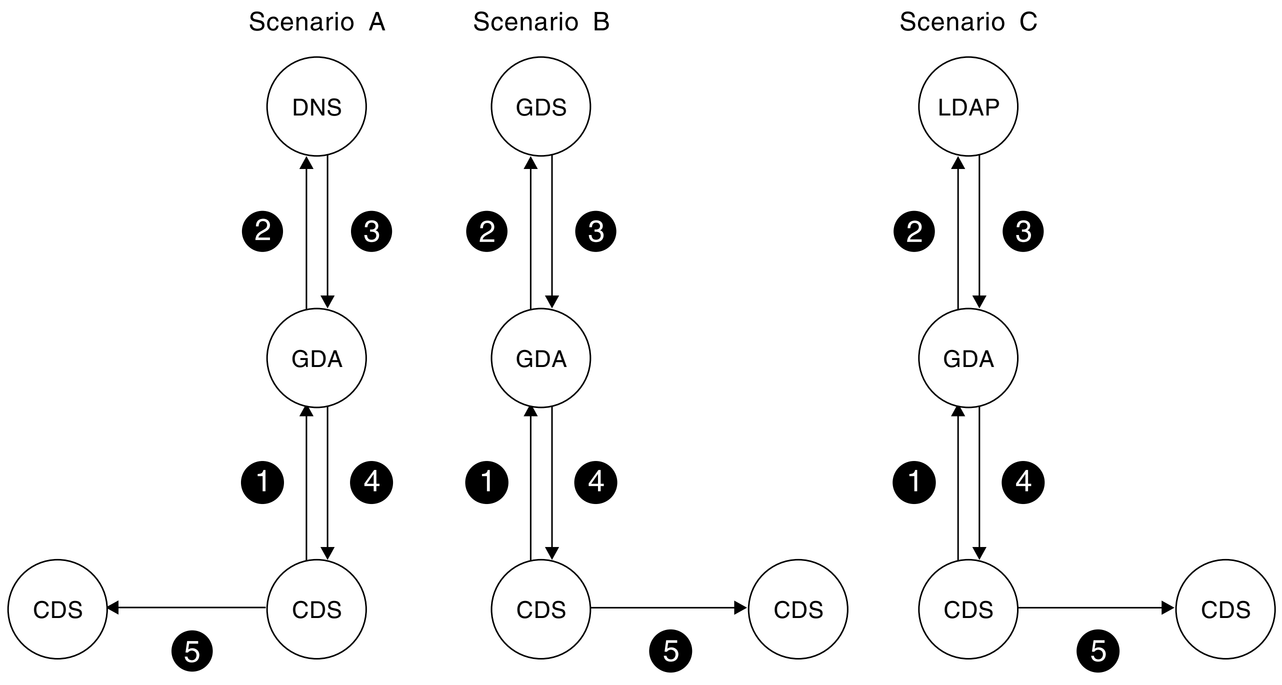
The **GDA** is the DCE component that makes cell interoperation possible. The GDA enables CDS to access a name in another cell by way of any of the global naming environments (GDS, DNS, or LDAP). The GDA is an independent process that can exist on a system separate from a CDS server. CDS needs to be able to contact at least one GDA to participate in the global naming environment.

Figure 18 shows how the GDA helps CDS locate binding information of names in another 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 (GDS, DNS, or LDAP server) for more information about the name.

The GDA returns information that enables CDS in the original cell 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 in one registered in GDS (Scenario B) or in one registered in LDAP (Scenario C).

Note: OS/390 does not support the use of GDS for looking up names.

The GDA decides which global service to use based on the syntax of the name. Later sections of this chapter discuss name syntaxes in detail.



The GDA helps CDS resolve names
A. in another cell that is registered in DNS
B. in another cell that is registered in GDS
C. in another cell that is registered in LDAP .

Figure 18. Interaction of CDS, GDAs, and Global Directory Services

How Cells Determine 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 names** and **cell-relative**, or **local**, names. This section introduces the concept of global and local names. Later sections discuss CDS, GDS, and DNS names in detail.

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).

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 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 equal sign (=). The items that are separated by an equal sign are multiple Attribute Value Assertions (AVAs). See the *OSF DCE GDS Administration Guide and Reference* for more information about AVAs. The first part of a pair is an abbreviation that indicates a type of information. Some common abbreviations are C (country), O (organization), OU (organization unit), and CN (common name).

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.

$$\overbrace{\text{/.../C=US/O=XYZ/OU=Portland}}^{\textit{Cell name}}\overbrace{\text{/subsys/PriceMax/price\_server1}}^{\textit{CDS name}}$$

Figure 19. Example of a Global Name

As the figure 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.

If the global cell name supplied matches the name of the local cell name, the **CDS name** portion is resolved in the local cell.

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 next example shows the global name of a host at ABC Corporation. The global name of the company's cell, **/.../abc.com**, exists in DNS.

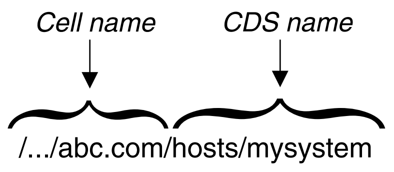


Figure 20. Global Name of a Cell in DNS

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 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. For example, 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 naming conventions required for the interaction of local and global directory services might at first seem confusing. In an environment where references to names outside of the local cell are necessary, a few simple guidelines can help make the conventions easy to remember and use:

- Know your cell name.
- Know whether a name 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 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 (names with a **/.:** prefix) are intended only for interactive use and should not be stored. (If a local name is referred to 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 Filenames

When referring to pathnames 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**, one level down from the root of the cell namespace. So, for example, 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
```

A Closer Look at DCE Names

The rest of this chapter takes an in-depth look at the different kinds of names that make up the DCE namespace. An appendix to this book contains further details about valid characters and naming conventions in CDS, GDS, and DNS names.

Cell Directory Service Names

Every cell contains at least one system running a **CDS server**. A CDS server stores and maintains names and handles requests to create, change, 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, conceptually similar to the directories in hierarchical file systems of many operating systems (for example, UNIX), 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 21 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.

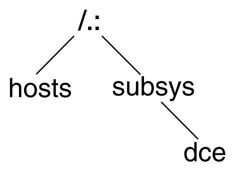


Figure 21. Sample CDS Namespace Hierarchy

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 the preceding figure contains an entry for a host whose simple name is **bargle**. The CDS full name of 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.

Global Directory Service Names

The operation of GDS is similar to that of CDS, but some important differences exist in the structure of names and the ways they can be looked up. Like CDS, GDS has a server process that provides access to and management of names. This process is called a **directory system agent** (DSA). The combined knowledge of all DSAs that participate in the same global directory service implementation is called the **directory information base** (**DIB**). This collective knowledge is viewed as a single global directory consisting of many entries.

Information exists in the global directory in the form of a rooted hierarchy called a **directory information tree** (DIT). The DIT is similar to a CDS namespace. However, unlike a namespace, which has no inherent rules regarding structure and content, the GDS hierarchy is influenced by a set of rules called a **schema**. Every X.500 DSA must define a standard schema to which all of the entries in its portion of the DIB conform.

Although the X.500 standard does not mandate a specific schema, it does make general recommendations based largely on existing X.400 standards for electronic mail. For example, countries and organizations should be named close to the root of the DIT; people, applications, and devices should be named further down in the hierarchy. GDS supplies a default schema that complies with these recommendations.

Every GDS entry has a **distinguished name**, which uniquely and unambiguously identifies that entry. The distinguished name consists of a sequence of valid **relative distinguished names** (RDNs). Each RDN consists of an assertion of the type and value of an attribute at a particular position in the DIT. **Attribute types** indicate the nature of the information stored in the attribute value. A pair consisting of an attribute type and value is known as an **Attribute Value Assertion** (AVA). RDNs can have multiple AVAs. For example, the distinguished name

```
/C=us/O=osf/OU=branch1/CN=nollman,OU=doc-team
```

consists of four RDNs. The final RDN consists of two AVAs.

Figure 22 on page 169 illustrates the concepts of RDNs and distinguished names, and shows how they relate to the DIT. The figure shows:

- A DIT consisting of a hierarchy of schema-defined attribute types
- RDNs that result from assertions of an attribute type and value
- Distinguished names that result from a concatenation of the RDNs

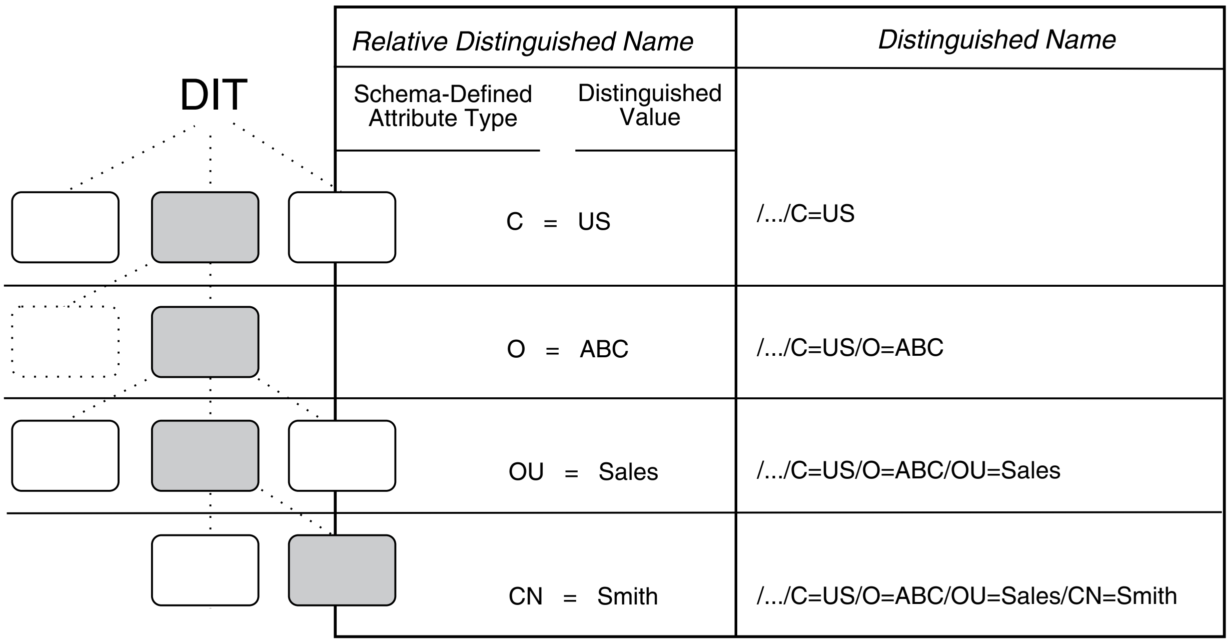


Figure 22. RDNs and Distinguished Names

The shaded boxes in the DIT represent the entries that are named in the column labeled Relative Distinguished Name. The schema dictates that countries are named directly below the root, followed by organizations, organization units, and names of users. Each attribute value that makes up an RDN (and thus a distinguished name) is called a **distinguished value**.

As the rightmost column in the figure illustrates, the distinguished name of the entry at each level of the DIT is a concatenation of RDNs from the root of the global directory to that entry's level. The lowest entry in the hierarchy, **/.../C=US/O=ABC/OU=Sales/CN=Smith**, represents the name of a user, John Smith, who works in the sales division of ABC Company, an organization in the United States. The abbreviated attribute type labels stand for country (C), organization (O), organization unit (OU), and common name (CN).

Note that the figure shows the global DCE convention for distinguished names. Each distinguished name starts with the representation of the global root (**/...**). Attribute types and values are separated by equal signs, and RDNs are separated by slashes.

These conventions for specifying names are not followed by all X.500 implementations. In addition, these conventions are only used at the GDS administration interface level. Internally, distinguished names are specified in other ways.

The structure of GDS names points out another important difference between GDS and CDS. A CDS name is distinct from its attributes; consists of a string of directory names ending with the simple name of the entry. In contrast, a GDS name consists solely of a series of attribute types and their values.

Figure 23 on page 170 illustrates this difference in the construction of CDS and GDS names. The CDS full name **/.:/Admin/Personnel/Employee_DB** is the complete directory specification of an entry with the simple name **Employee_DB**. Attributes and their values are not a part of the CDS full name. The GDS distinguished name **/.../C=US/O=ABC/OU=Sales** is a concatenation of attribute types and values, one from each level of a DIT schema.

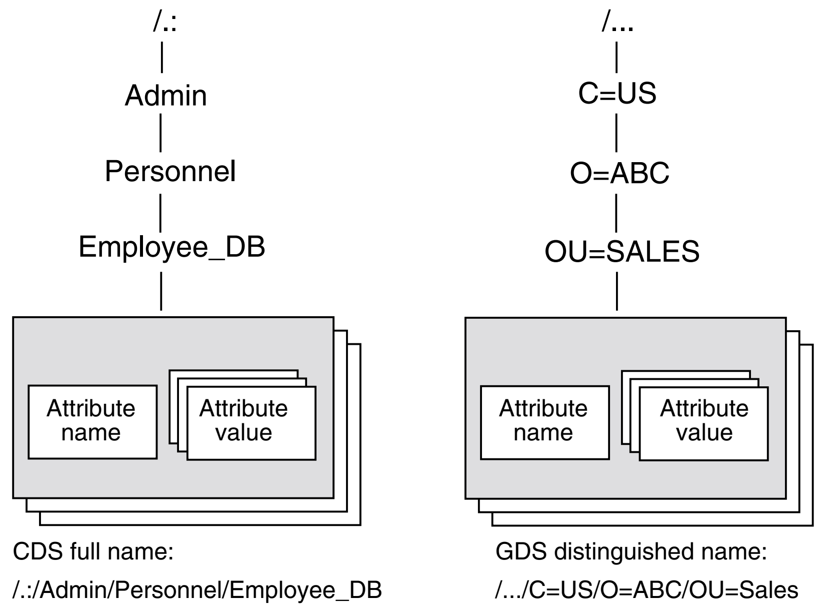


Figure 23. Comparison of CDS and GDS Names

GDS supports the ability to search for names by supplying the values of one or more attributes. This results in what is called **descriptive naming**; in a sense, users can describe the name they are looking for. Although the search capability is valuable, it can be expensive and time-consuming, so GDS allows users to restrict the scope of a search. Support for the search operation is limited to the GDS environment.

Domain Name System Names

The DCE naming environment supports the version of DNS based on Internet Request for Comments (RFC) 1034 and RFC 1035. Many networks currently use DNS primarily as a name service for hostnames. 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 *OS/390 DCE: Planning* 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 24 on page 171 shows a sample portion of the BIND namespace. The double quotation marks ("") 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 (do not consist of pairs of attribute types and values).

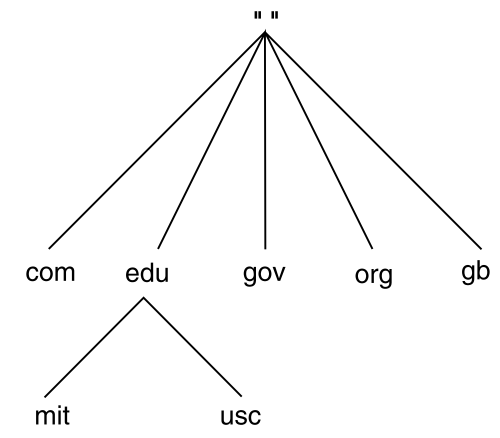


Figure 24. Sample Portion of the BIND Namespace

A DNS name consists of a string of hierarchical names 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 complete DNS name between two slashes. For example, a cell whose DNS name is **ai.mit.edu** might contain a directory whose CDS name is **/.:/profiles**. Users would 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.

For example, 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**. The following example illustrates the parts of a global DCE principal name:

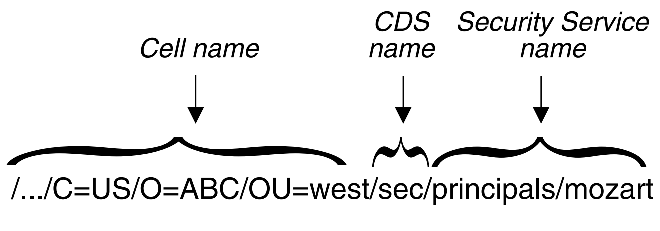


Figure 25. Parts of a Global Principal Name

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 stored in the Security Service database.

Another service that uses junctions is DFS. 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 following example illustrates the parts of a global DCE file name:

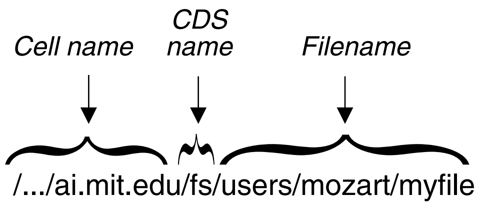


Figure 26. Parts of a Global DCE File Name

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.

Note: DFS has its own set of publications. Consult those manuals for more information.

The DCE namespace is thus 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.

Chapter 19. Cell Directory Service Concepts

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 consistent. Names also can 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.

How the Cell Directory Service Works

Operation of the Cell Directory Service involves several major participants:

- Client applications
- Servers
- Clerks
- Advertiser
- 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, change, 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 change names. One clerk can work on behalf of many client applications.

The clerk caches, or 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.

The CDS Advertiser solicits and advertises the names of all CDS server databases (which will be discussed later in this section). The CDS Advertiser must run on every DCE node.

Figure 27 shows a sample configuration of CDS clerks and servers on a 9-node local area network (LAN). Every node is a clerk, and CDS servers run on two selected nodes.

Note: In this figure, all nodes in the LAN must run the CDS Advertiser.

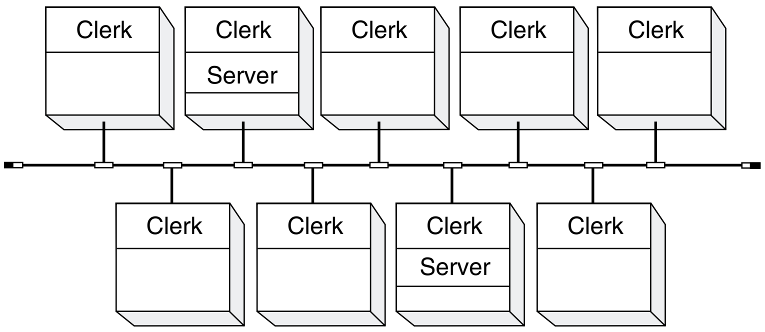


Figure 27. CDS Clerks and Servers on a LAN

Every CDS server has a database called a **clearinghouse** in which it stores names and other CDS data. Using the clearinghouse, the CDS server adds, changes, deletes, and retrieves data on behalf of client applications. Although more than one clearinghouse can exist at a server node, it is not recommended as a usual configuration.

Figure 28 on page 175 shows the interaction between a CDS client, clerk, server, and clearinghouse during a simple lookup.

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.

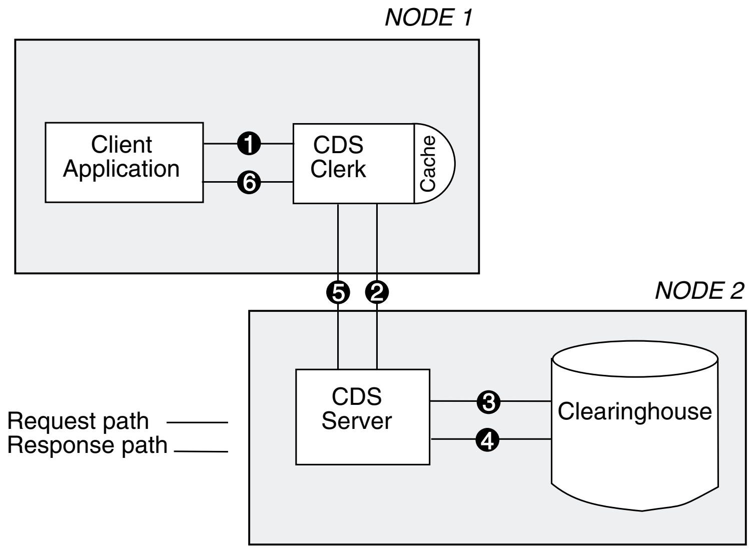


Figure 28. A Sample CDS Lookup

Replicas and Their Contents

Directories are the units by which you 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
- 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 Chapter 28, “Restructuring a Namespace” on page 239 for complete information on how to redesignate the master replica of a directory.)

The master replica is the only directly changeable replica of a directory. CDS can create, change, and delete information in a master replica. Because it is modifiable, 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, change, or delete names in read-only replicas; it simply updates them with changes made to the master replica.

Replicas can contain three kinds of entries:

- Object entries
- Soft links
- 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 values, 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 is used to 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 *OS/390 DCE: 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 *OS/390 DCE: Command Reference* for more information on how 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. CDS itself updates and manages clearinghouse object entries when necessary. They do not require any external management except in rare problem-solving situations.

Soft Links

A **soft link** is a pointer that provides an alternative name for an object entry, directory, or other soft link in the cell namespace. You can do minor restructuring of a cell namespace by creating soft links that point from an existing name to a new name. Soft links also can 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 period of time that you specify. If the name that a soft link points to is deleted, CDS waits for the soft link to expire, and then deletes it the next time the directory is updated.

You should use soft links carefully. Do 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 keep track of and manage.

Child Pointers

A **child pointer** connects a directory to another directory immediately beneath it in a cell 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.

Putting It All Together

To summarize, a cell namespace consists of a complete set of names 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 picture 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 server.

The following figure 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 possible in a replica. Usually, a clearinghouse contains many replicas, and a replica contains many entries.

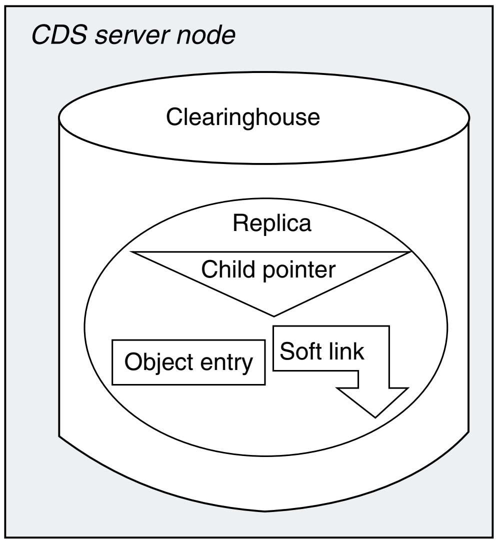


Figure 29. Components of a CDS Server Node

CDS Advertiser

In OS/390 DCE, the **CDS Advertiser**, also known as the **Advertisement and Solicitation** daemon, performs the following functions:

- Acquires and stores information and status of other CDS servers and clearinghouses on the network
- Broadcasts the status and information about the clearinghouses that are served by the CDS server on the local host, if any
- Creates a cache and loads it from a cache database file.

The CDS Advertiser is usually started when the DCEKERN address space is started. You can also start it using the MODIFY DCEKERN command.

CDS Advertiser and Clerk in OSF DCE

In the OSF implementation of DCE, the CDS Advertiser *forks* a CDS Clerk for every DCE client. Figure 30 on page 178 shows the interaction between a CDS client, CDS Clerk, and CDS Advertiser in the OSF model.

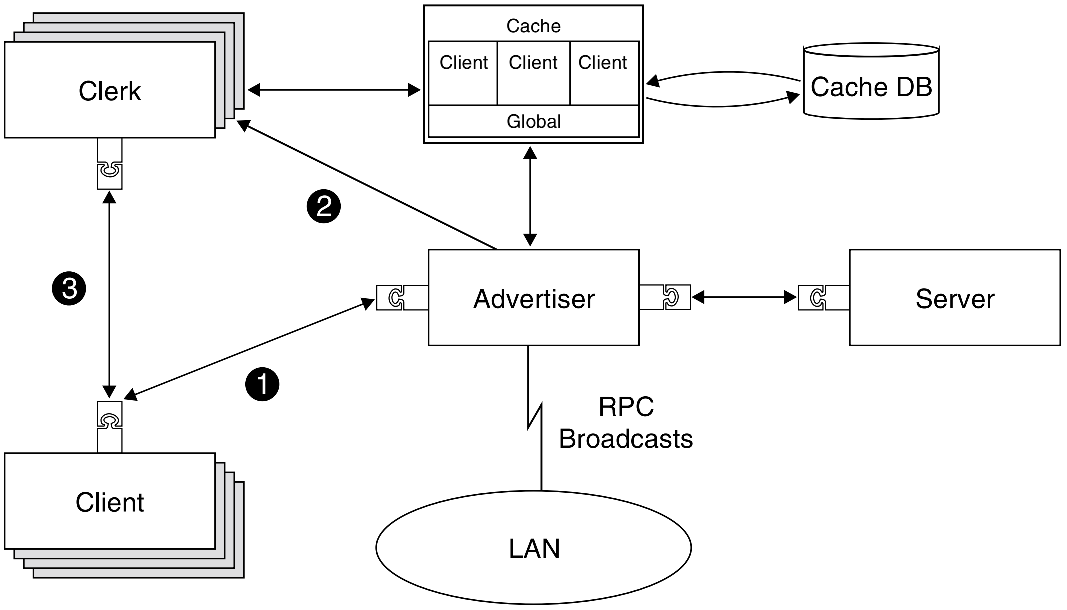


Figure 30. Interaction between the CDS Client, Clerk, and Advertiser in OSF DCE

In this model, the following events occur when a client submits a request to CDS:

1 A client forwards its request to the Advertiser. The Advertiser determines if a CDS Clerk has previously been forked for that client. If a CDS Clerk already exists, the CDS Advertiser simply advises the client of the socket (port) at which the Clerk is listening.

2 If not, the Advertiser forks a CDS Clerk and advises the client of the socket which the particular Clerk uses to “listen” to the client.

3 The client connects to and sends all requests to the CDS Clerk.

The Clerk stores the response to a query in its cache. This information is also periodically saved to disk by the CDS Advertiser.

The CDS Clerk runs as **root** that has absolute access rights to the security credentials cache files of all clients.

If there is no activity between the client and the DCE Clerk for a significant amount of time, socket communication between the client and the DCE Clerk is terminated and DCE Clerk daemon stops.

CDS Advertiser and Clerk in OS/390 DCE

There is a fundamental difference in the implementation of the CDS Clerk between the OSF and OS/390 DCE models.

In OS/390 DCE, the CDS Advertiser does not fork a new CDS Clerk for each new CDS Client. Rather, there is only one CDS Clerk that processes all requests from all DCE clients. The CDS Clerk and the CDS Advertiser exist as distinct tasks in the DCEKERN address space and are started when DCEKERN is brought up. Figure 31 on page 179 shows the interaction between the CDS Client, Clerk, and Advertiser in OS/390 DCE.

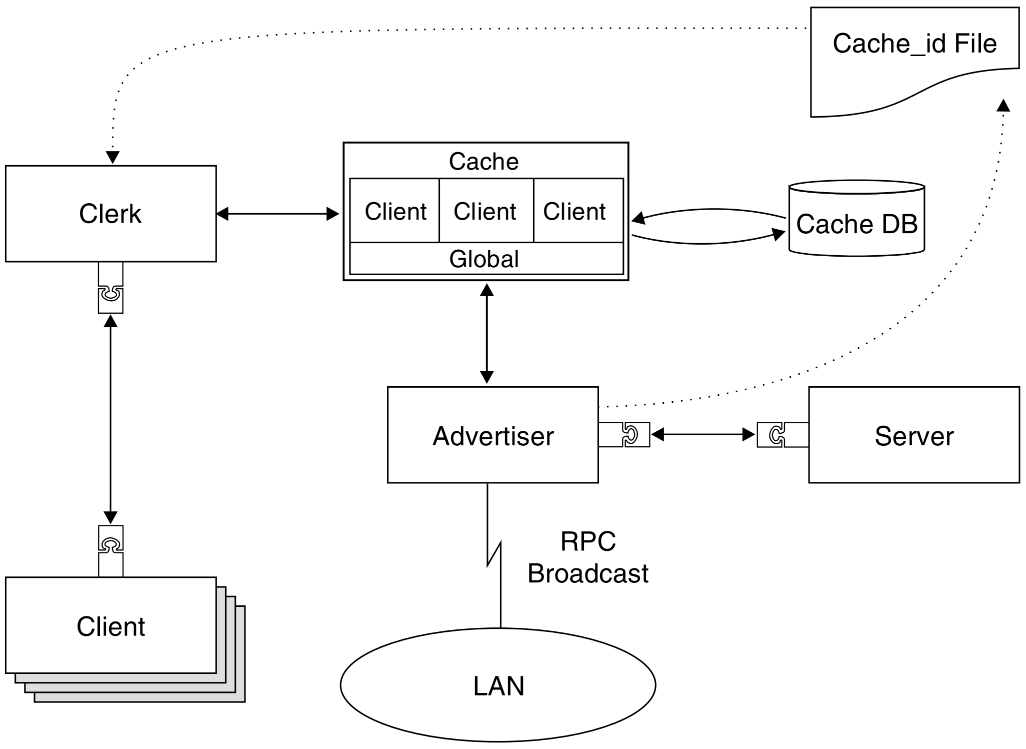


Figure 31. Interaction between the CDS Client, Clerk, and Advertiser in OS/390 DCE

Unlike the OSF implementation, the CDS Clerk is a *long-living* process; that is, it continues to run even if there is no activity between a client and the CDS Clerk.

Because only one *long-living* CDS Clerk exists to process all incoming requests, the CDS Clerk *listens* on a well-known socket for client requests in OS/390 DCE.

The CDS Clerk, which runs as root, has **read** permission to all security credentials cache files of clients.

Security in the 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. So, 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.

CDS servers themselves must be authenticated principals for two reasons:

- To prove to clients that they are trustworthy
- To prove to each other that they have the permission to change 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 instead 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 change 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 changes 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 change that directory.

The CDS permissions are read, write, insert, delete, test, control, and administer. Each has a slightly different meaning depending on the kind of name it is associated with, but, 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.

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 a common ACL entry for unauthenticated users determines whether the user has the permissions to perform the requested action. See Part 8, “DCE Security Service” on page 293 for details on creating ACLs for unauthenticated users.

Protecting the CDS Cache File

This section discusses how the CDS cache file is secured from unauthorized users.

In the OSF implementation of DCE, a trusted channel is established between each DCE Clerk and his client through local sockets that are protected by POSIX access controls. In OS/390 DCE, where one DCE Clerk services all clients, each client must still be able to delegate its authority to the single DCE Clerk in a secure manner. In this case, the local socket is not protected by any POSIX access controls. The Clerk must be able to determine the identity of the clients in a secure and trusted manner.

Each client exports its login context to the Clerk. The login context contains additional data: the user ID of the client on the local host. The Clerk verifies that this login context really belongs to the client by querying the client's identity in a trusted manner. If the user ID of the client that is obtained by the Clerk does not match the user ID in the login context, it closes the communication with the client.

The CDS Clerk cache contains different types of data. These include CDS names, attributes, and addresses of clearinghouses and servers. The cache is populated by any read operation.

Retrieving data from the cache is performed by the CDS Clerk. Access to the CDS Clerk Cache data is restricted so that only clients who first request and receive the data are the only ones who can later retrieve it from the cache. In accessing the cache, the CDS Clerk assumes the identity (and therefore the login context) of the user who made the request to the Clerk. Access to the cached information is allowed or denied based on the identity of the requester. For example, user A cannot retrieve information from the cache (through the CDS Clerk) that user B had earlier requested and received.

Any unauthenticated user can retrieve data from the cache that has been previously requested by another unauthenticated user.

Conversion between ASCII and EBCDIC in OS/390 DCE

The OSF implementation of CDS was intended for ASCII platforms, such as UNIX workstations. Because OS/390 DCE runs on an EBCDIC platform, there are compatibility problems when the OS/390 DCE CDS daemons attempt to communicate with other DCE servers running on ASCII-based hosts.

To avoid these problems, CDS processes in OS/390 DCE appear to the distributed computing environment as if they are running on an ASCII platform. Data from the local code page is translated to code page 819 while the data is sent on the wire. Code page **819** is the IBM implementation of the International Organization for Standardization (ISO) ASCII **8859/1** code page.

String character data in the OS/390 DCE Clerk's cache is stored in the local EBCDIC code page, and the opaque form of the data is in ASCII code page. When the opaque data needs to be manipulated by CDS, it is converted into EBCDIC code.

Conversion between code pages is done internally by CDS on OS/390 DCE.

Cell Directory Service User Interface

CDS has several *entities* that can be managed by means of user interfaces that are provided in DCE. A CDS entity is any individually manageable piece of the CDS software. 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 usually managed through the client application that creates them.

The **dcecp** program provides many commands for managing CDS entities. Chapter 22, “Managing the DCE Directory Service” on page 193 contains information about these commands.

Another user interface is the **cdscp** program, which is available on all clerks and servers. It is an interface that accepts commands targeted for specific entities and directory replicas. (An entity is any individually manageable piece of the CDS software.)

Note: 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 in CDS servers, clerks, and replicas are described in the next several chapters.

In addition to the **dcecp** program and the **cdscp** program, 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 an ACL editor such as the **dcecp acl** object or the **acl_edit** program that is supplied with the Security Service. RPC application programmers can create server entries, groups, and configuration profiles in the cell namespace with the **dcecp** or **rpccp** programs. (See other parts of this guide, for details on how to use these interfaces.)

Chapter 20. How the Cell Directory Service Looks Up Names

This chapter illustrates the relationship between a name and the physical resource it describes, and 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. Note that the figures in this chapter do not reflect the actual structure of a typical DCE cell namespace. For simplicity, the figures show fewer directories and directory levels.

Translating from Names to Resources

Just as 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 illustrates the relationship between CDS names and the physical resources they describe.

Figure 32 on page 184 shows three directories and their contents in a logical namespace, and how replicas of those directories are physically located 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.

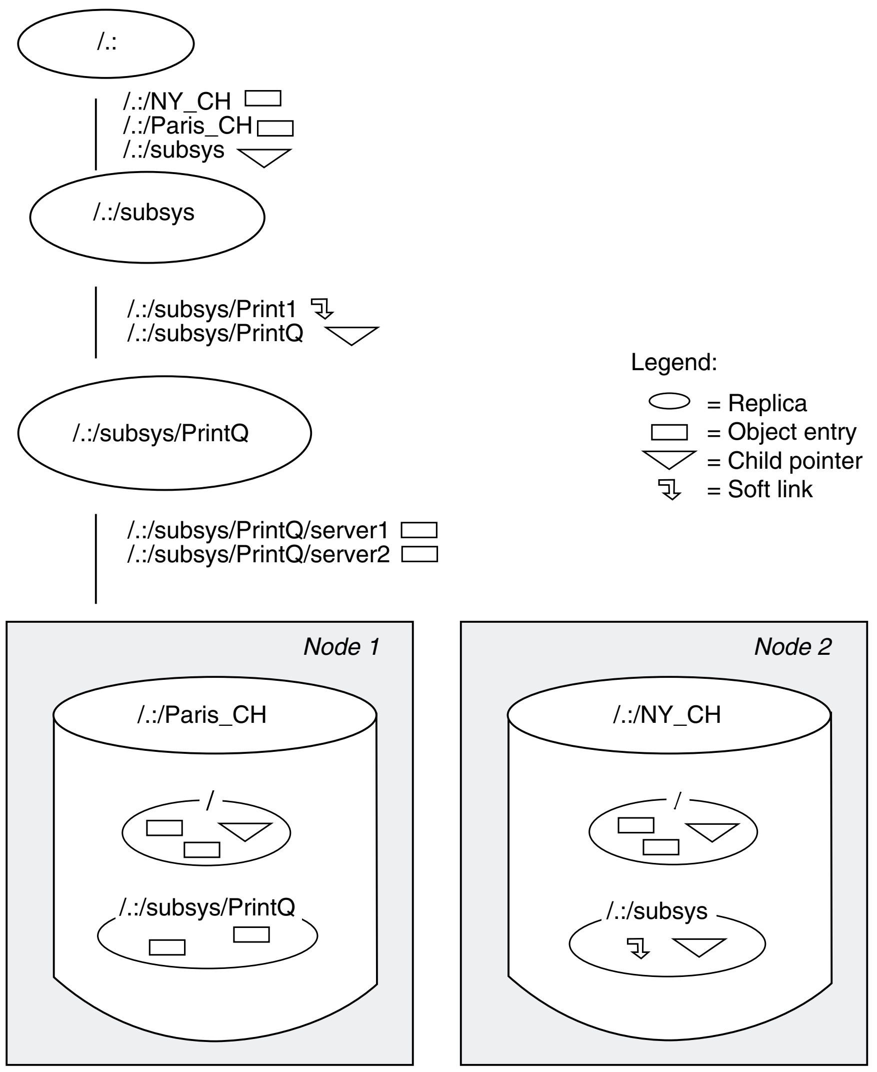


Figure 32. Logical and Physical Views of a Namespace

To discover the physical location of a resource, CDS looks up an attribute associated with its name. The next three figures illustrate the connection between various kinds of CDS names and the resources they describe. The figures are based on the namespace in the first figure in this chapter. All of the names in the figures are in the same cell namespace, as evidenced by the use of the **/.:** prefix to represent the cell root. For information about name resolution across multiple cells, see Chapter 29, "Managing Intercell Naming" on page 251.

Figure 33 on page 185 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: the clearinghouse. CDS creates the object entry automatically when you create and name the clearinghouse.

The figure shows two clearinghouse object entries: **/.:/Paris_CH**, which points to the clearinghouse named **/.:/Paris_CH** on Node 1, and **/.:/NY_CH**, which points to the clearinghouse named **/.:/NY_CH** on Node 2. Each clearinghouse object entry has an attribute called **CDS_CHLastAddress**, whose **Tower** subattribute contains RPC binding information that CDS uses to contact the node where the clearinghouse resides.

Refer to Appendix E, “Object Identifier Files” on page 535 for a list of CDS attributes and their descriptions.

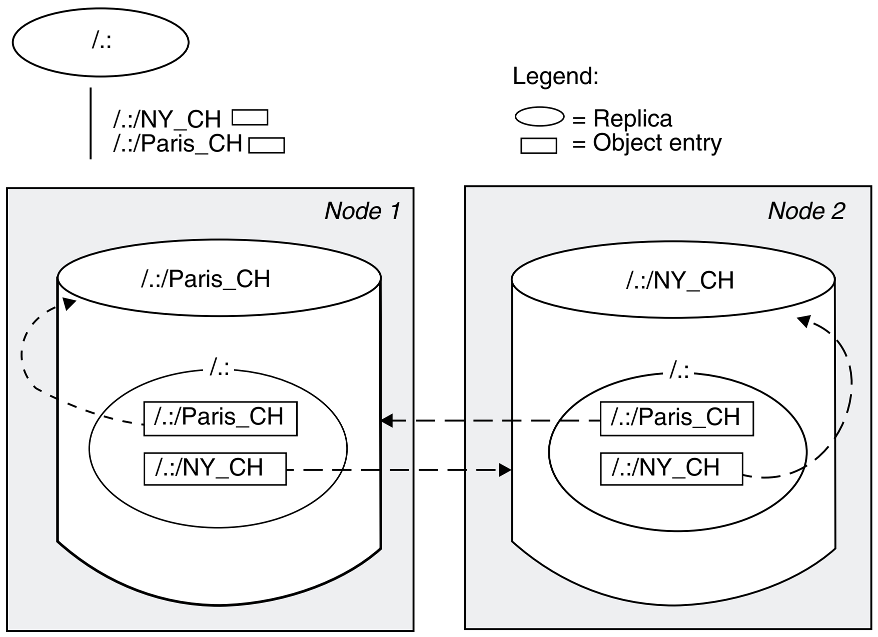


Figure 33. Clearinghouse Object Entries and Clearinghouses

Figure 34 on page 186 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 that the link points to: an object entry named **/.:/subsys/PrintQ/server1**. The object entry describes a print server machine 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 an attribute called **CDS_CHLastAddress**, whose **Tower** subattribute contains RPC binding information that enables the PrintQ application to contact the print server machine.

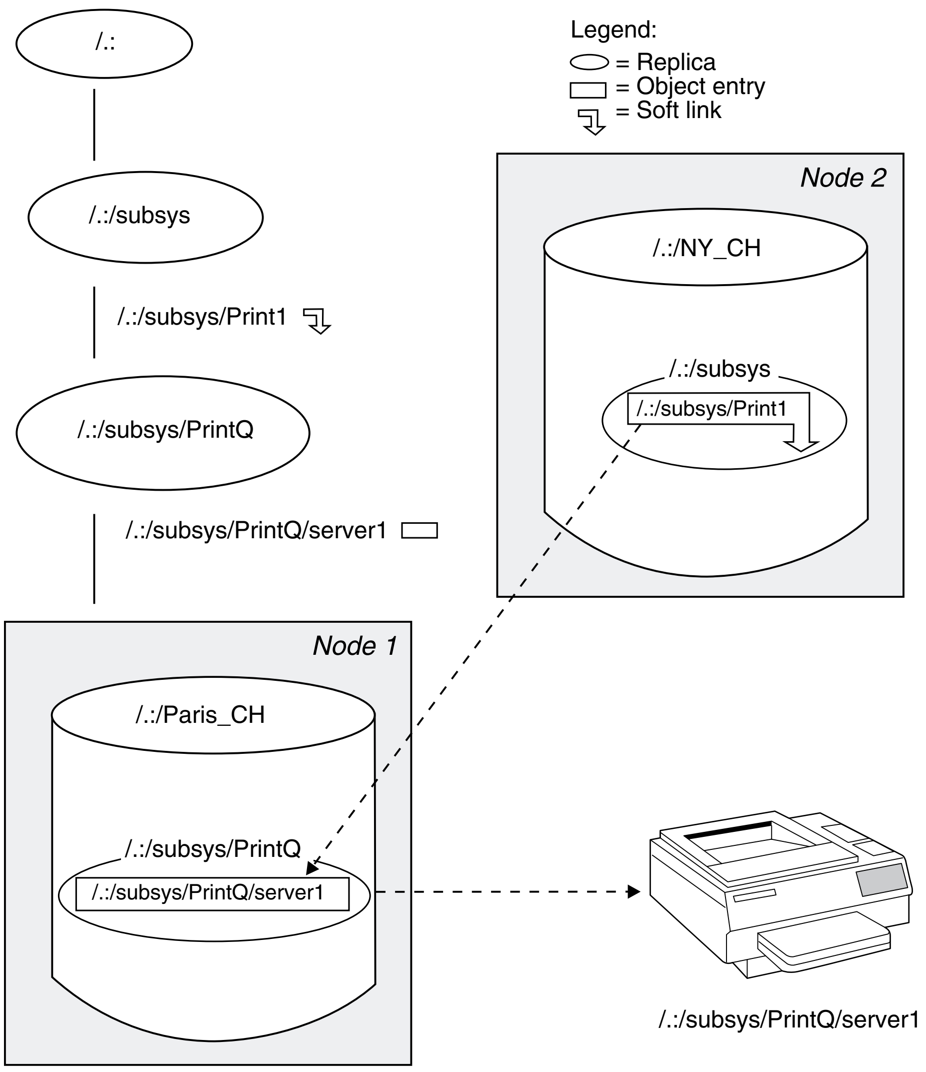


Figure 34. A Soft Link and Its Resolution

Figure 35 on page 187 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.

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 would usually be the case, the **CDS_Replicas** attribute lists all of the clearinghouses containing a replica of the directory.

You can use the DCE control program (**dcecp**) **directory show** command with the **-replica** and **clearinghouse** options to display this attribute.

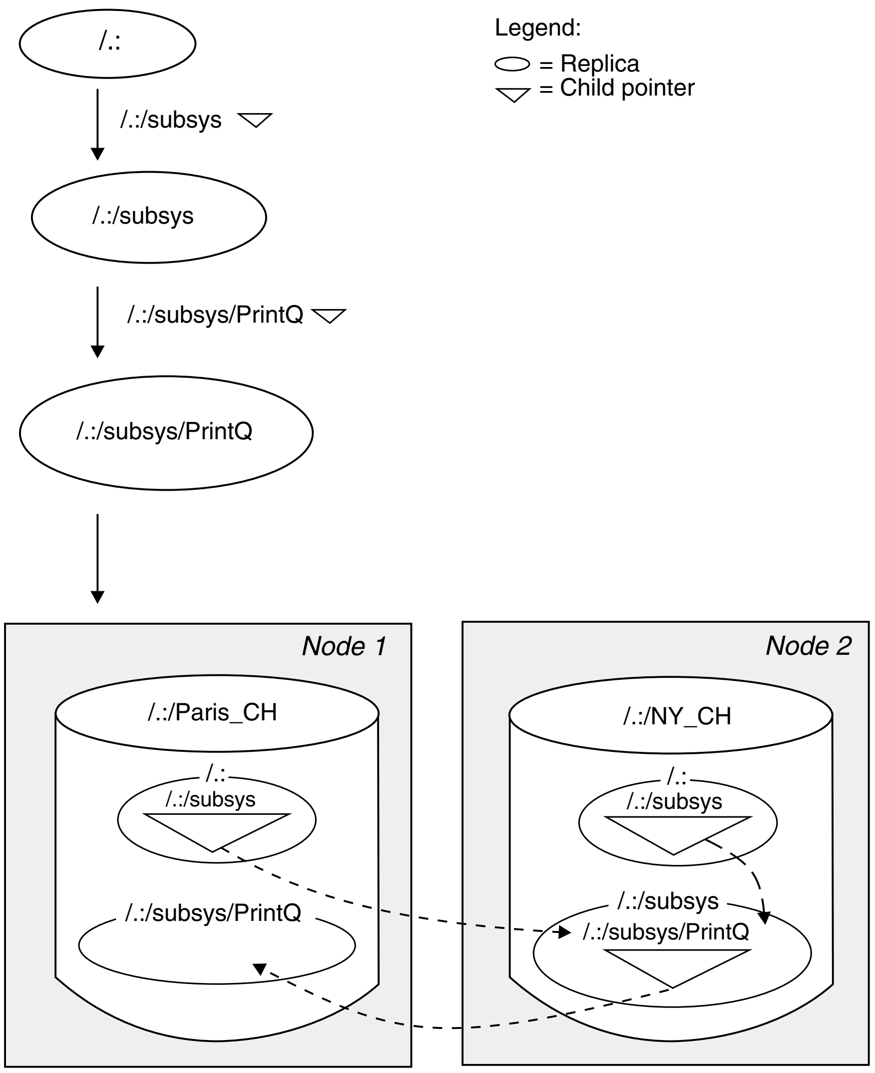


Figure 35. Child Pointers and Directories

How the Cell Directory Service Finds Names

As the previous figures illustrate, CDS finds information about the physical location of a resource by looking up one or more attributes associated with its name. First, the clerk must know how to find the name. If a name does not yet exist in the clerk's cache, the clerk must know of at least one CDS server to contact in search of the name.

The clerk can learn about CDS servers and their locations in any of three ways:

- Through the **solicitation and advertisement protocol**
- During a regular lookup
- By response to the **cdscache create** command

The Solicitation and Advertisement Protocol

Clerks and servers on the same local area network (LAN) communicate 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 data about the cell that the server belongs to, the server's network address, and the clearinghouse it manages. The CDS Advertiser on the local host learns about servers by listening for these advertisements on the LAN. The CDS Advertiser on the local host also sends out solicitation messages, which request advertisements, at startup.

Lookups

During a lookup, if a clearinghouse does not contain a name that the clerk is searching for, the server managing that clearinghouse gives the clerk as much data as it can about where else 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 use 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 therefore cannot learn about the server from advertisements on a LAN.

Figure 36 on page 189 is 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.

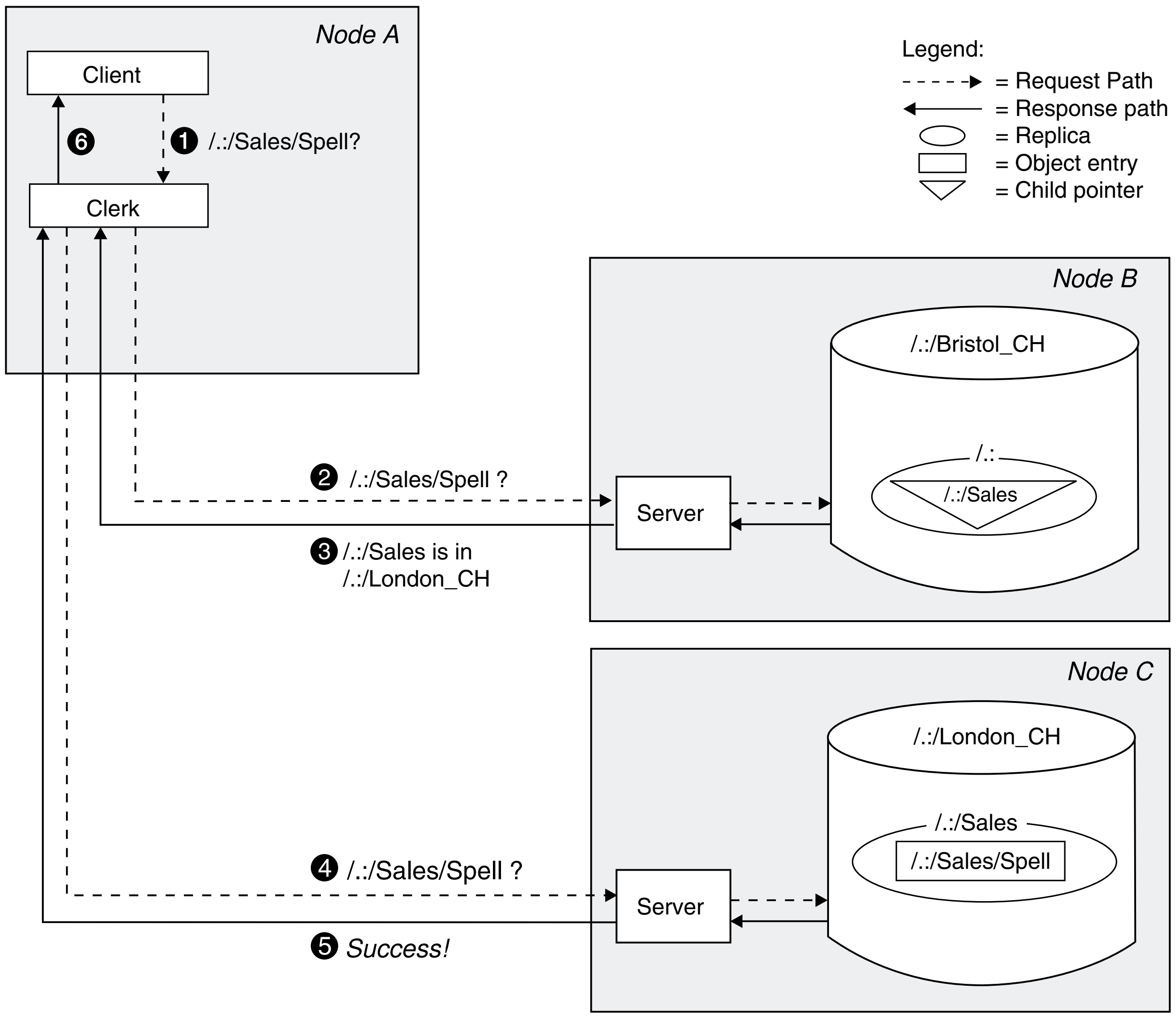


Figure 36. How the Clerk Finds a Name

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 like the one illustrated in Figure 36 do not 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 somewhere the directory path is broken or a clearinghouse is unreachable, a clerk might not be able to find a name.

Chapter 21. How the Cell Directory Service Updates Data

Once names exist in the namespace, users who have the appropriate access can make changes to the data associated with the names. Any addition, modification, or deletion of CDS data initially happens in only one replica: the master replica. This chapter introduces the main methods by which CDS keeps other replicas consistent: **update propagation** and the **skulk** operation. It also describes two timestamps that help to ensure consistency in CDS data. By understanding the concepts in this chapter, you can more effectively plan the content and replication of directories in your namespace.

Update Propagation

An update propagation is an immediate attempt to apply one change to all replicas of a directory in which the change was just made. Its main benefit is that it delivers each change in an efficient and timely way. Unlike a skulk operation, however, update propagation does not guarantee that the change is made in all replicas. If a particular replica is not available, the update propagation does not fail; the change simply 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 altogether, by adjusting a directory attribute called **CDS_Convergence**. Convergence also affects the frequency of skulks on a directory. See Chapter 25, “Managing CDS Directories” on page 219 for details on viewing and changing a directory's convergence.

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. These maintenance functions are:

- Removing soft links that have expired (you can specify an expiration time when you create a soft link)
- Maintaining child pointers, which includes removing pointers to directories that were deleted
- Removing information about deleted replicas.

CDS skulks each directory individually. During a skulk, CDS collects all changes made to the master replica since the last successful skulk, and then disseminates the changes to all 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 attempts the skulk again until it is successfully completed.

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

  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.

How Timestamps Help Keep Data Consistent

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 **Creation Timestamp** (**CTS**) to everything 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 generated on different nodes.

During propagation of a new name or a 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 **Update Timestamp** (**UTS**) reflects the most recent change 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. The *OS/390 DCE: Command Reference* describes those timestamps and how CDS uses them.

Note: In OS/390 DCE, the CDS client daemons (CDS Advertiser and Clerk) use the time provided by DTS. If DTS is not running, CDS client daemons use the time provided by the OS/390 system clock along with an offset from GMT.

Chapter 22. Managing the DCE Directory Service

The DCE control program (**dcecp**) provides most of the commands you need to manage CDS. This chapter describes the CDS entities that the DCE control program permits you to manage and summarizes the available commands for managing these entities.

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**). This chapter provides some information that you need to run the **cdscp** program and notes the operations for which the program's commands are required.

For detailed descriptions of the **dcecp** and **cdscp** program commands discussed in this chapter, see the Command Reference.

Using the dcecp Program

Chapter 7, “DCE Control Program Introduction” on page 45 introduced you to the **dcecp** program and its command syntax, so this chapter does not repeat this information. This chapter describes the commands that the **dcecp** program supplies specifically for managing CDS.

CDS Managed Objects

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 that appears 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.

dcecp Command Operations for CDS

The **dcecp** program commands for managing CDS perform the operations shown in Table 6.

Table 6 (Page 1 of 2). The dcecp Command Operations for CDS

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.

Table 6 (Page 2 of 2). The dcecp Command Operations for CDS

Command Operations	Definitions
create	Creates an object in the cell namespace. The object type can be a directory, object entry, soft link, clearinghouse, CDS cache, 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.
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 of the CDS objects contained in a 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.
remove	Removes a child directory from a parent in the cell namespace.
rename	Changes the name of a specified object.
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 section in the *OS/390 DCE: Command Reference*. 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.

Using the cdscp Program

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.

Starting and Exiting

To start the **cdscp** program, enter:

```
$ cdscp
```

To exit from the **cdscp** program, enter:

```
cdscp> quit
```

To read a file of the **cdscp** commands while inside the control program, enter:

```
cdscp> do filename
```

The cdscp Program Commands

Table 7 lists the **cdscp** program commands that you use to manage CDS clerks, servers, directory replicas, and cells.

If you are using a file or script to issue **cdscp** commands and you want to include a comment as part of a **cdscp** command, use a # or ! as the first character. Any string following these characters is ignored (including the comment character itself). The character cannot be preceded by the back slash escape character. The only way to include it in your entry name or attribute value is to enclose the simple name or attribute value in a pair of single or double quotation marks. Comments can be used in commands that are entered from the command line or as part of a **do** file.

Table 7. The cdscp Program Commands with No Equivalent dcecp Command

Commands	Definitions
disable clerk	Stops the execution of a CDS clerk.
set cdscp confidence	Permits you to set the confidence level of CDS clerk calls.
set cdscp preferred clearinghouse	Permits you to set a preferred clearinghouse for **cdscp** commands.
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 cdscp preferred clearinghouse	Permits you to see the preferred clearinghouse for **cdscp** commands.
show clerk	Displays the attributes of a CDS clerk.

For detailed descriptions of all of the **cdscp** program commands, see the **cdscp** section in the *OS/390 DCE: Command Reference*.

Chapter 23. Controlling Access to CDS Names

This chapter discusses management information on CDS authorization.

Overview of DCE Authorization for the Cell Directory Service

CDS authorization lets you control user access to the following CDS components:

- Names stored in the namespace, including clearinghouses, directories, object entries, soft links, and child pointers
- Running 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 **ACL 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.

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 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 Command Reference for detailed information on the commands.

The remainder of this chapter describes DCE authorization as it applies specifically to CDS.

Before you try to create or change permissions to CDS names, clerks, or servers, read Chapter 34, "Using Access Control Lists" on page 307 for complete information on the DCE authorization mechanism and how to use the ACL Editor.

ACL Types Supported by the Cell Directory Service

CDS supports the following DCE ACL types:

- **Object ACL**
  You can use the Object ACL type to grant permissions to any CDS name (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**
  The Initial Object Creation ACL type applies only to CDS directory names. Use this ACL type to grant permissions specifically to a directory's future contents (soft links, application-defined object entries, child pointers, and clearinghouse object entries). The permissions you grant 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 extended to names that already exist in the directory.

To edit an Initial Object Creation ACL, specify the **-io** option of the **dcecp** program's **acl modify** command.

- **Initial Container Creation ACL**
  The Initial Container Creation ACL type applies only to CDS directory names. Use this ACL type to grant permissions to a directory that automatically extend (default) to all child directories that you may later create under that directory. The permissions you grant 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 extended to child directories that already exist.

  To edit an Initial Container Creation ACL, specify the **-ic** option of the **dcecp** program's **acl modify** command.

How Permissions Propagate to CDS Directories and Their Contents

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.

For example, suppose you just created a new directory named **/.:/sales**. If you create an ACL entry of the Object ACL type granting 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.

If you create a second ACL entry of the Initial Object Creation ACL type granting **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 (unless prohibited by explicit ACL modification after their creation).

If you create a third ACL entry of the Initial Container Creation ACL type, also granting **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
- Perform all of the above operations on all child directories that may later be created under the **/.:/sales** directory.

See Chapter 34, “Using Access Control Lists” on page 307 for complete information on default ACLs.

Access Control List Entry Types

You use ACL entry types to specify the category of principal for which the ACL is created. CDS supports the DCE ACL entry types described in Table 8 on page 199.

Table 8. ACL Entry Types Supported by CDS

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 all principals whose credentials were authenticated within the local cell, unless they are specifically named in ACL entries of type **user** or are members of an authorization group named in ACL entries of type **group**.
foreign_user	Specifies an ACL entry for an individual principal whose credentials were authenticated in another named cell.
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 named in ACL entries of type **group**.
any_other	Specifies an ACL entry for a user (either authenticated or unauthenticated) not otherwise covered by any of the preceding ACL entry types.
mask_obj	Specifies an ACL entry containing a mask that is ANDed together to form the permissions of any principals, whose credentials are either authenticated or unauthenticated.
unauthenticated	Defines the maximum permissions for any ACL entry specifying a principal whose credentials were not authenticated.
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** and 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 another named cell.
foreign_group_delegate	Specifies an ACL entry for an intermediary that acts for the members of an authorization group in another named cell.
foreign_other_delegate	Specifies an ACL entry for an intermediary that acts for authenticated principals in another named 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 another named cell who are not named by an ACL entry of any other type for intermediaries of authenticated principals or groups.

DCE Permissions Supported by the Cell Directory Service

CDS supports the following DCE permissions: read (r), write (w), insert (i), delete (d), test (t), control (c), and administer (a). 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:

- **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 (for use with directory entries only) allows a principal to create new names in a directory.
- **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 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 change the ACL entries associated with a name. (Note that **read** permission is also necessary for changing a CDS entry's ACLs; otherwise, **dcecp** and **acl_edit** will not be able to bind to the entry.) Control permission is automatically granted to the creator of a CDS name.
- **Administer** permission (for use with directory entries only) allows a principal to enter CDS commands that control the replication of directories.

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 saying 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 appropriate for the type of name created. For example, a principal creating an object entry is granted read, write delete, test, and control permission to the object entry. A principal creating a directory is granted read, write, insert, delete, test, control, and administer permission to the directory.

Note: Unlike the security mechanisms enforced by most other file directory systems, CDS does not require a principal to have access to all intermediate elements in the pathname (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 (**/.:**).

Controlling Access to CDS Clerk and Server Management Operations

CDS authorization lets you control the use of CDS commands that involve local management operations on CDS clerks and servers. Principal names for each clerk and server are stored in the security namespace. An object entry containing 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 ACL that contains ACL entries specifying the principals allowed to perform these operations. Unlike the ACLs associated with names in the namespace, the ACLs 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 permission 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 in to the system as the machine principal is permitted to run 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 change this scheme and grant permission to particular clerks and servers on behalf of other individual principals or authorization groups.

When editing an ACL associated with a CDS clerk or server entity, use the **dcecp** program's **acl modify** command with the **-change** option. For example, to change the permissions for the user **michaels** in the ACL that is associated with the CDS clerk on node **orion**, enter:

```
dcecp> acl modify /.:/hosts/orion/cds-clerk -change {user michaels rw}
```

Note: Keep in mind that clerks and servers are also represented by entries in the namespace. To edit an ACL associated with the namespace entry for a CDS clerk or server, you must include the **-entry** option, as well as the **-change** option, in the **acl modify** command line. For a detailed instructions on how to modify an ACL on the CDS entry for a DCE resource, see "Editing ACLs on Cell Directory Service Names" on page 203.

Control Program Commands and Required Permissions

Table 9 lists all the **dcecp** commands that operate on CDS objects and the permissions a principal must have to run the commands.

Table 9 (Page 1 of 3). The dcecp Program Commands and Required Permissions

Commands	Required Permission
cell show	Read permission to several directories in the CDS namespace.
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.

Table 9 (Page 2 of 3). The dcecp Program Commands and Required Permissions

Commands	Required Permission
directory list	Read permission to the directory whose contents you want to list.
directory modify	Write permission to the directory for which you want to add (**-add** option) or change (**-change** option) the attribute or attribute value. Delete permission to the directory for which you want to remove the attribute or attribute value (**-remove** option).
directory remove	Write permission to the parent directory of the child pointer (**-member** option) you want to remove.
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 wildcard directory name, you also need read 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. 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, UID=0) 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 catalog	No special permissions are needed.
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 lose knowledge of the clearinghouse.
clearinghouse show	Read permission to the clearinghouse whose attributes you want to list. If you specify a wildcard 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.

Table 9 (Page 3 of 3). The dcecp Program Commands and Required Permissions

Commands	Required Permission
server disable	Write permission to the server execution object.
server show	Read permission to the applicable server configuration or server execution objects.

Table 10 lists commands for operating on CDS objects that are only provided by the CDS control program (**cdscp**), along with their execute permissions.

Table 10. The cdscp Commands and Required Permissions

Commands	Required Permissions
disable clerk	Write permission to the clerk.
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 clerk	Read permission to the clerk.

Editing ACLs on Cell Directory Service Names

To edit an ACL that is associated with an entry in the CDS namespace for a child directory, 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 pathname. In some cases, a pathname 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.

For example, to edit the permissions in the Object ACL that is associated with a CDS entry for a clearinghouse named **/.:/Paris1_CH**, 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, enter this 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}
```

How CDS Servers Gain Access to the Namespace

CDS servers require permission to the cell root directory and to lower-level directories, to successfully run the following **dcecp** commands:

- **clearinghouse create**
- **directory create** (for directories and replicas)
- **directory delete** (for directories and replicas)
- **directory synchronize**

To automate the process of granting all CDS servers the permissions 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 grants full permission (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 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.

Setting Up Access Control in a New Namespace

You should plan a consistent access control policy and be ready to put it into place 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
- Establishing maximum permissions for unauthenticated principals.

Adding Members to the Namespace Authorization Group

To facilitate the management of 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 extends 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 **group add** command to add the principal names of other individuals in your organization who 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 Part 8, "DCE Security Service" on page 293 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 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 (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.

Chapter 24. Managing Clerks, Servers, and Clearinghouses

CDS clerks, servers, and clearinghouses are initially created and started as part of CDS clerk and server configuration. Thereafter, clerk and server processes are created and started with a series of commands run 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.

This chapter explains how to perform clerk, server, and clearinghouse management tasks.

Monitoring Clerk, Server, and Clearinghouse Counters

Every clerk, server, and clearinghouse maintains a set of attributes called counters to keep track of 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 being generated on your network. Clerk, server, and clearinghouse counters are fully described in the *OS/390 DCE: Command Reference*.

Displaying Clerk Counters

Use the CDS control program (**cdscp**) **show clerk** command to display the current counter values for a clerk. For example, to display the current values of all attributes associated with a clerk, enter:

```
cdscp> show clerk
```

Displaying Server Counters

Use the **cdscp** program **show server** command to display the current counter values for a server.

For example, to display the current values of all attributes associated with a server, enter:

```
cdscp> show server
```

Displaying Clearinghouse Counters

Use the DCE control program (**dcecp**) **clearinghouse show** command with the **-counters** option to display the current counter values for a specified clearinghouse. For example, the following command displays the current values of all attributes associated with the remote clearinghouse **/.:/Paris1_CH**:

```
dcecp> clearinghouse show  /.:/Paris1_CH  -counters
```

Monitoring Clerk Communication with Specific Clearinghouses

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 to 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, you use the **dcecp** program's **cdscache show** command with the **-clearinghouse** option. For example, to display the cached clearinghouse counters that are maintained by the local clerk for the **/.:/NY1_CH** clearinghouse, enter the following:

```
dcecp> cdscache show /.:/NY1_CH -clearinghouse
```

The following command example displays the cached clearinghouse counter values for all the clearinghouses used by the local clerk:

```
dcecp> cdscache show  /.:/*  -clearinghouse
```

Displaying the Contents of a Clearinghouse

Use the **dcecp** program **clearinghouse show** command with the **-clearinghouse** option and specify the **CDS_CHDirectories** attribute to display the directory names of all the directories stored in a particular clearinghouse.

For example, to display the names of the directories stored in the clearinghouse **/.:/Chicago2_CH**, enter:

```
dcecp> clearinghouse show /.:/Chicago2_CH
```

See Chapter 26, "Viewing the Structure and Contents of a Namespace" on page 227 for more examples on displaying clearinghouse information.

Forcing the Clearinghouse to Checkpoint to Disk

Under usual 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 entering the **dcecp clearinghouse initiate** command with the **checkpoint** option.

For example, to checkpoint the clearinghouse **/.:/Boston3_CH** from memory to disk, you enter the following command:

```
dcecp> clearinghouse initiate /.:/Boston3_CH -checkpoint
```

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 active clerk or server processes to be suspended.

You can disable local CDS clerks and servers by using the **disable clerk** and **disable server** commands of **cdscp**. Servers that are *not* running on OS/390 can be disabled by using the **dcecp server disable** command.

Disabling a Clerk

To disable the clerk that is on the local node, enter the following command:

```
cdscp> disable clerk
```

Disabling a Server

To disable the server that is on the local node, enter the following command:

```
cdscp> disable server
```

To disable one or more servers running on a host other than an OS/390 host, enter the following command:

```
dcecp> server disable server_name_list -interface interface_id_list
```

where *server_name_list* is one or more servers to be disabled and *interface_id_list* is one or more RPC interfaces to be disabled.

Stopping the CDS Advertiser and CDS Clerk

You may occasionally have to stop the CDS Advertiser and CDS Clerk running on a particular host, for example, when you need to perform diagnostic work. The CDS Advertiser and CDS Clerk daemons run as tasks in the DCEKERN address space and can be stopped by using the MODIFY operator command. For example, to stop the CDS Advertiser on the OS/390 host, enter:

```
MODIFY DCEKERN,stop cdsadv
```

To stop the CDS Clerk:

```
MODIFY DCEKERN,stop cdsclerk
```

For more details on the MODIFY operator command, see "The MODIFY DCEKERN Operator Command" on page 35.

Alternatively, you can also disable the CDS Advertiser and the CDS Clerk by using the **disable clerk** command from CDSCP:

```
cdscp> disable clerk
```

Note: This single command stops both the CDS Advertiser and the CDS Clerk daemons.

Starting and Stopping the CDS Daemon

The CDS daemon (**cdsd**) runs as a task in its own address space and can be started and stopped by using the MODIFY operator command. For example, to stop the CDS daemon on the OS/390 host, enter:

```
MODIFY DCEKERN,stop cdsd
```

For more details on the MODIFY operator command, see "The MODIFY DCEKERN Operator Command" on page 35.

Alternatively, you can also disable the CDS daemon by using the **disable server** command from CDSCP:

```
cdscp> disable server
```

Restarting the CDS Advertiser and CDS Clerk

When either the CDS Advertiser or CDS Clerk ends, it must be restarted in the correct sequence. The CDS Advertiser must be started before the CDS Clerk because the CDS Advertiser has to create and load the cache database file before the CDS Clerk can start accessing it.

To restart the CDS Clerk:

1. Log in to the host where the Clerk is to be restarted with the appropriate privileges to start the CDS Advertiser and CDS Clerk.

2. Determine if the DCE host daemon is running on the host. You can use the MODIFY command to determine the status of this process. If DCED is not running, start it using the MODIFY command.
3. If SECD is configured on this host, determine if it is running. You can use the MODIFY command to determine the status of this process. If SECD is not running, start it using the MODIFY command.
4. Use the MODIFY command to start the CDS Advertiser and CDS Clerk.

Automatic Restart after Abnormal Termination

In OS/390 DCE, the CDS Advertiser and CDS Clerk daemons are automatically restarted after abnormal ending, using these conditions:

- If the CDS Clerk ends abnormally, the CDS Advertiser is also stopped. The CDS Advertiser is then restarted, followed by the CDS Clerk.
- If the CDS Advertiser ends abnormally, it is restarted without ending the CDS Clerk.
- In certain error situations occurring when the CDS Advertiser is being restarted, if the CDS Clerk also needs to be stopped and restarted, the Advertiser starts the Clerk automatically.

Removing Stale Cache Entries

Stale entries in the CDS cache refer to binding information that points to non-existent servers. This may happen for a variety of reasons. The server may have ended abnormally, or has been moved to a different location. The host system where the server was running may have shut down. Communication in the network may have failed.

DCE clients that attempt to make remote procedure calls to servers using the stale server information from the cache will receive a communications error message.

This situation can be addressed in any of the following two ways:

- Have the DCE client log in to DCE again. Doing so creates a new credentials cache file, the opaque form of login context. Because the key to access the client cache is the new credentials cache file, the user cannot access the previous CDS cache entries.
- Using the CDS control program, set the **cdscp confidence** to either **medium** or **high** and then run the **show object** subcommand using the offending server entry as the argument to this subcommand.

  If the **cdscp confidence** is set either to **medium** or **high**, the CDS Clerk obtains the information directly from the CDS server and then refreshes the information contained in the CDS cache. Note that because the **set confidence** subcommand is valid only for the current CDSCP session, the **set confidence** and **show object** subcommands must be run within the same CDSCP session.

  For example, if the client receives a communications error message after attempting to contact an application server whose namespace entry is **/.:/servers/appl**, you can run the following subcommands from the CDS control program:

  ```
  cdscp> set cdscp confidence = medium
  cdscp> show object /.:/servers/appl
  ```

  Note: The **set cdscp confidence** command can be placed in the **$HOME/.cdscpinit** file.

Recovering from a Corrupted CDS Cache

If the CDS cache file becomes corrupted, you can generate a new cache by performing the following steps:

1. Stop the DCEKERN address space. This will also stop the OS/390 DCE daemons.
2. Delete the corrupted cache files. There are two files to delete, the version file, **/opt/dcelocal/var/adm/directory/cds/cds_cache.version**, and the actual cache file, **/opt/dcelocal/var/adm/directory/cds/cds_cache.***nnnnnnnnnn*, where *nnnnnnnnnn* is a 10-digit number.
3. Restart DCEKERN using the **-nodce** option. This option starts the DCEKERN address space without starting the configured OS/390 DCE daemons.
4. Using the MODIFY command, restart the following OS/390 DCE daemons in the following order: DCE host daemon (**dced**), security daemon (**secd**), CDS Advertiser (**cdsadv**), CDS Clerk (**cdsclerk**), and CDS daemon (**cdsd**).
5. Run the **define cached server** command of the CDS control program, which creates knowledge of a CDS server in the cache. For example:

   ```
   cdscp define cached server host1 tower ncadg_ip_udp:9.21.22.10
   ```

   or

   ```
   dcecp -c cdscache create host1 -binding ncadg_ip_udp:9.21.22.10
   ```

6. Start the DTS daemon and any remaining configured daemons (using the MODIFY DCEKERN command).

If Cache Size Is Changed

If you change the size of the CDS cache by passing a different cache size to the CDS Advertiser, the cache will not be loaded from disk. Instead, the CDS Advertiser will create a new instance of the cache. In this case, you will have to follow Steps 1 to 6 that are outlined in “Recovering from a Corrupted CDS Cache,” that is, you have to run the **cdscp define cached server** command.

Preserving a Clearinghouse across a Server System Upgrade

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, follow this procedure:

1. Make sure that you stop the clerk and server.
2. Before you perform the system upgrade, back up these CDS files:

   cds_attributes
   cds_files
   <*clearinghouse-name*>**_ch.checkpoint***nnnnnnnnnn*
   <*clearinghouse-name*>**_ch.tlog***nnnnnnnnnn*
   <*clearinghouse-name*>**_ch.version**
   cds_cache.*nnnnnnnnnn*
   cds_cache.version
   cds_cache.wan

   In the above file names, *nnnnnnnnnn* is the version number of the CDS database. The files are in **/opt/dcelocal/etc**, **/opt/dcelocal/var/directory/cds/**, and **/opt/dcelocal/var/adm/directory/cds/**.

3. Perform the system upgrade.
4. Restore all the files that you backed up in Step 2 on page 211.
5. Follow the procedure, described earlier in this chapter, for restarting a server. When the server process starts, it automatically locates the appropriate restored files and starts all clearinghouses on the system.

Recovering from a cdsd File System Full Condition

With the addition of the CDS Server to OS/390 DCE, some HFS space issues arise. The *OS/390 DCE: Planning* book recommends that you establish a separate file space in HFS for the CDS daemon (**cdsd**) to store its data. The mountpoint is **opt/dcelocal/var/directory/cds**, and the recommended size for the file space is 30MB.

The CDS Server for OS/390 DCE has been enhanced beyond the OSF capabilities to provide some automatic attempts to recover if this file system becomes full. Each attempt to checkpoint a clearinghouse to the file system first ensures that there is sufficient space to checkpoint. In addition, once per hour, a threshold-level check of the file system space occurs. The space available is checked when the CDS Server is first started, to determine if existing clearinghouses should be placed into read-only mode or brought up in read-write mode.

When verifying that there is space for the new checkpoint file in the file system, the CDS Server first ensures that the checkpoint will not fill up **opt/dcelocal/var/directory/cds**. The “full” level is considered to be 98 percent full. If there is insufficient space in this file system, the CDS Server next checks the space levels in a user-defined alternate file space, which defaults to **/tmp**. The CDS Server attempts to copy the existing checkpoint file to the alternate file space, removing it from **opt/dcelocal/var/directory/cds**. This allows room for the new checkpoint file to be stored in **opt/dcelocal/var/directory/cds**. The CDS Server only copies to the alternate file space if the addition of the checkpoint file keeps the alternate file space less than 65 percent full. If the CDS Server cannot find space in either location to allow checkpointing, the clearinghouse is changed to read-only mode, and only read operations are permitted against the clearinghouse. If the clearinghouse is in read-only mode when a checkpoint is requested, and if space has been made available, the clearinghouse changes back to read-write mode and a checkpoint occurs.

During the hourly threshold-level checking, several conditions are checked, to determine if a checkpoint of the clearinghouse is needed. The CDS Server for OS/390 DCE adds a similar space check as described above. If the other condition checking recommends a checkpoint, but there is not enough space for the checkpoint, the recommendation to checkpoint is canceled. During this checking, the clearinghouse is changed to read-only mode if there is not enough space. Once the clearinghouse is in read-only mode, if space becomes available, the clearinghouse is changed back to read-write mode.

The environment variable to use to define the alternate file space is _EUV_CDSD_ALT_FILESPACE. It should be added to the CDS Server's envar file in **/opt/dcelocal/home/cdsd**. This variable must specify a fully-qualified path name. For example:

```
_EUV_CDSD_ALT_FILESPACE=/home
```

where *home* is the target file space. If this variable is not specified, **/tmp** is used as the alternate file space.

In both the checkpoint verification step and the threshold-checking step, various messages may appear on the console. Here are the messages and the actions to be taken as a result:

- The user threshold message:

  ```
  CDS server detects space has reached user threshold of nn percent.
  ```

This message can occur during either the hourly threshold checking or regular checkpointing. You can set the percentage full level where you want to be notified of space conditions on **opt/dcelocal/var/directory/cds**. This level is set by defining the environment variable _EUV_CDSD_FILESPACE_THRESHOLD in the envar file in **/opt/dcelocal/home/cdsd**. If this variable is not specified, you start getting notifications when the percentage full level reaches 85 percent. You can turn off this level of threshold checking by specifying the _EUV_CDSD_FILESPACE_THRESHOLD variable to be 100 percent. Here is how to turn off this threshold level:

```
_EUV_CDSD_FILESPACE_THRESHOLD=100
```

When this message is received, it is a signal that you should examine the space situation for **opt/dcelocal/var/directory/cds**. If you have taken advantage of the capability to define the file system with secondary extents, it is possible that you have enough space available. Verify the number of extents the file system has taken and the space available on the volume where extents are allocated.

If the information found indicates that you have sufficient space, the message can be ignored. You may choose to set the _EUV_CDSD_FILESPACE_THRESHOLD variable to a higher level or turn off this level of checking completely. Regardless of the setting of this environment variable, checking is still done for the 98 percent full threshold level.

However, if the information found indicates a potential space problem, you can take some actions to prevent the clearinghouses from going into read-only mode. First, using a superuser ID, examine the **opt/dcelocal/var/directory/cds** file space. Verify that there are no unnecessary files in this directory. Unless someone has inadvertently copied a file into this directory, you should not find any extraneous files. The files that should be in the directory are:

- **cds_files**
- **server.acl.v1.dat**
- **adm** (a directory)
- *cellname#clearinghouse_name.version* for each clearinghouse
- *cellname#clearinghouse_name.checkpoint.v000000000n* for each clearinghouse
- *cellname#clearinghouse_name.tlog.v000000000n* for each clearinghouse

If you find extraneous files in **opt/dcelocal/var/directory/cds**, they should be deleted, or moved to another directory. If you do not find extraneous files, you should next examine your defined alternate file space. If none is defined, examine **/tmp**. Use the shell command **df -Pt /***directoryname* to determine how full the alternate file space is. In order for the CDS Server to be able to make use of the alternate file space, the currently-used space plus the size of the current clearinghouse checkpoint file in **opt/dcelocal/var/directory/cds** cannot exceed 65 percent of the total space available for the alternate file space. A good guideline is for the alternate file space to have around 50 percent of its space available. Again, this can be accomplished by deleting items from the alternate file space or by moving them to another directory.

If you find you cannot make space available in either file system, you should schedule an outage for this CDS Server, and create a larger file space for the **opt/dcelocal/var/directory/cds** directory. To create a larger file space, do these steps:

1. Use either the **cdscp disable server** command in the shell or the **f dcekern,stop cdsd** console command to stop **cdsd**
2. Copy the files in the current **opt/dcelocal/var/directory/cds** file space to the new, larger file space
3. Mount the new file space at **opt/dcelocal/var/directory/cds**
4. Restart the CDS Server

- “Insufficient space” message:

  ```
  CDS server detects insufficient space for checkpoint.
  ```

  This message is followed by:

  ```
  Changing database to read-only state.
  Clearinghouse: clearinghouse_name
  ```

  This message can be received during either the threshold-checking step or the checkpoint space-verification step. When this message is received, **opt/dcelocal/var/directory/cds** has no room for another checkpoint. If the CDS Server was unable to make use of the alternate file space, either because it would be more than 65 percent full by copying the checkpoint file, or because removing the checkpoint file from **opt/dcelocal/var/directory/cds** still does not allow room for a new checkpoint, then this message appears.

  The same steps listed above for checking the space situation in **opt/dcelocal/var/directory/cds** and the alternate file space should be taken in this situation. If it has been possible to clear enough space, a checkpoint can be forced by using the dcecp command, **dcecp clearinghouse initiate** *clearinghouse_name* **-checkpoint**. If space is available, a checkpoint is taken and the clearinghouse is changed back to read-write mode.

- “Server unable to retrieve data” messages:

  ```
  CDS server unable to retrieve file system data.
  ```

  or

  ```
  CDS server unable to retrieve alternate file system data.
  ```

  Either of these messages is followed by:

  ```
  Changing database to read-only state.
  Clearinghouse: clearinghouse_name
  ```

  Either of these messages can be received during either the threshold-checking step or the checkpoint space-verification step. If one of these messages is received, then either there is some internal error in the CDS Server or there is a problem with the HFS file system. Check for other messages regarding potential HFS file system problems. If none are found, and HFS seems to be operating correctly, call your service representative for assistance.

- “Server unable to restore file” message:

  ```
  CDS server unable to restore copy of checkpoint file.
  ```

  This message is followed by:

  ```
  Changing database to read-only state.
  Clearinghouse: clearinghouse_name
  ```

  This message is received during either the threshold-checking step or the checkpoint space-verification step. The CDS Server found that it needed to move a copy of the checkpoint file to the alternate file space. It successfully moved the file to the alternate file space and deleted the checkpoint file from **var/directory/cds**. However, it was unable to establish a symbolic link for the file, and when it attempted to move the copy of the checkpoint file that was in the alternate file space back into **var/directory/cds**, it could not complete the move. The checkpoint file remains in the alternate file space. The steps for recovering space listed above should be followed. If it is possible to recover space and complete a new checkpoint successfully, the copy of the old checkpoint file in the alternate file space can be deleted. If it is necessary to stop and restart the CDS Server, the checkpoint file that is in the alternate file space must be manually copied or linked to **var/directory/cds**.

  To create a symbolic link, do the following from within the **opt/dcelocal/var/directory/cds** directory:

  ```
  ln -s tmp_checkpoint_copy_name checkpoint_file_name
  ```

Look at the other files related to this clearinghouse in this directory for the format to use for *checkpoint_file_name*. This name includes the cell name, while the *tmp_checkpoint_copy_name* does not.

- "Now able to checkpoint" message:

  ```
  CDS server now able to checkpoint.
  ```

  This message is followed by:

  ```
  Changing database to read-write state.
  Clearinghouse: clearinghouse_name
  ```

  This message is received during either the threshold-checking step or the checkpoint space-verification step. This message is received when the space has become available for checkpointing and the clearinghouse is being changed back to read-write mode.

- "Using alternate file space" message

  ```
  The CDS server is using alternate file space pathname
  to allow checkpointing.
  ```

  This message is received any time the CDS server uses the alternate file space to store a copy of a checkpoint file. It is informational and indicates that the alternate file space is being used.

- "Alternate file space cleanup" message

  ```
  CDS server successfully cleaned up alternate file space.
  ```

  This message is received when the CDS server has successfully completed a new checkpoint and has successfully removed the copy of the old checkpoint file from the alternate file space. It is informational and indicates that the alternate file space is no longer being used.

- "Unable to restore checkpoint file" message

  ```
  CDS server unable to restore copy of checkpoint file.
  Clearinghouse: clearinghouse_name
  ```

  The CDS Server used the alternate file space to save a copy of the current checkpoint file, but was unable to complete the operation. This message is received when the CDS Server fails to restore the copy back into **var/directory/cds**. The checkpoint file remains in the alternate file system. The steps described above for ensuring adequate space for checkpointing should be followed and a checkpoint of the clearinghouse should be done using the **dcecp initiate** command. This re-establishes the correct linkage to the checkpoint files. If the checkpoint fails or it is necessary to restart the CDS server before completing a checkpoint with **dcecp initiate**, it is necessary to manually link the checkpoint file, or manually copy the file back to **var/directory/cds**.

- "Unable to remove checkpoint file" message

  ```
  The CDS server is unable to remove the checkpoint copy in the
  alternate file system.
  File_name:  clearinghouse_copy_file_name
  ```

  The CDS Server used the alternate file space to save a copy of the current checkpoint file, but was unable to complete the operation. This message is received when the CDS Server successfully restores the copy back into **var/directory/cds**, but is unable to remove the copy of the old checkpoint file from the alternate file space. The file can be removed manually.

Reconfiguring a Secondary CDS Server After Deconfiguring

DCE users may experience problems when they configure an additional (secondary) CDSD on OS/390, then at some later time deconfigure and reconfigure the additional CDSD. Reconfiguring CDSD within a short time after deconfiguring may result in skulk failures, with a status code indicating a decryption integrity check failure or a status code indicating that CDS is unable to communicate with the CDS server.

The first status code is a “decryption integrity check” (0x1412901F) on the skulk, and often shows up in a probe, as an error message:

```
EUVR00338E RPC_CN_AUTH_VFY_CLIENT_REQ on server failed.
  Error text: Decryption integrity check fails (dce / krb).
```

This status is returned when the reconfiguration of the CDSD secondary server is attempted within the ticket expiration lifetime (default value is 10 hours) of the DCE principal for CDSD. In order to force daemons holding tickets to refresh the tickets, you must start and stop the CDS daemons (CDSD, clerk, and advertiser) on the system that is the master for the root directory of the namespace. If skulk failures persist on other machines in the cell, it may also be necessary to start and stop the CDS daemons on the other machines in the cell.

The second status code is “cannot communicate with any CDS server” (0x10d0a3ec) on the skulk. This status is returned because there is still an entry for the clearinghouse you are attempting to reconfigure in the **cdsclerk** cache on the machine that is the master for the root directory. In order to clear up the cache, stop the CDS daemons on that machine, manually delete the **cds_cache.*** files in the **/opt/dcelocal/var/adm/directory/cds** directory, and restart the CDS daemons. You can then manually attempt the skulk command (**cdscp set dir /.: to skulk**) or wait for the skulk to successfully complete on its own. If the machine that is the master for the root directory of your namespace is running AIX DCE 2.2 or higher, you may be able to clear the entry from the cache without stopping the daemons and removing the cache. Please refer to AIX DCE 2.2 documentation for more information.

If you attempt to deconfigure the secondary CDSD on OS/390, this attempt to deconfigure will probably fail with one of the status codes listed above on the **skulk** command and the “Directory must be empty” status code (0x10d0a3fc) on the **delete clearinghouse** command. Any further attempts to configure an additional CDSD on OS/390 will fail on the **create clearinghouse** command. The clearinghouse object is still present, but you cannot show the clearinghouse. In addition, a **cdscp show dir /.:** command indicates that there is a clearinghouse on OS/390 when there is not. In this case, you must manually clean up on the machine that is the master for the root directory of your namespace by doing these steps:

Note: Check the documentation for your specific platform for the process to use for recovery in this situation. The following recommended steps have been successfully used when the master CDSD for the root directory is on the IBM AIX platform, and the cell configuration is not complex.

1. Delete the clearinghouse object using **cdscp delete object**.
2. Delete the clearinghouse using **cdscp delete clearinghouse**. (This may fail, which is normal in this case.)
3. Remove the OS/390 clearinghouse. (Issue the **cdscp set directory to new epoch** command, excluding the OS/390 replica from the replica set.)
4. Issue a skulk command (**cdscp set dir /.: to skulk**).

See the *DCE Command Reference* (the one that corresponds to the platform where your master CDSD for the root directory resides) for command syntax of the cdscp commands listed above.

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.

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 221 for complete information on how to create a replica.

Using Operating System Backups

If you decide to use operating system backups, you only need to back up the server systems whose clearinghouses store master replicas of directories. Make sure you disable the servers on these systems by using the **disable server** command *before* you perform the backups.

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.

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, change, 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 and what modifications were made since that time.

Chapter 25. 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.

This chapter describes the process for creating directory hierarchies in the cell namespace and for other tasks related to managing directories.

Creating a Directory

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 lets you take advantage of default ACL entry propagation.

CDS cell configuration creates an initial hierarchy of directories under the root so that DCE components can fix 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 Appendix C, "The DCE Cell Namespace" on page 499 for more information on the initial hierarchy established by cell configuration.

Permissions for Creating a Directory

To create a directory, you need the following permissions:

- Insert permission to the parent of the new directory
- Write permission to the clearinghouse that stores the master replica of the new directory.
- The server principal for the server system where you enter the DCE control program (**dcecp**) **directory create** command must have read and insert permission to the parent directory of the new directory.

  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 **acl show** command on the parent directory to verify that the server principal has the appropriate permissions. See Part 8, "DCE Security Service" on page 293 for complete information on arguments to the **acl show** command.

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, CDS, by default, stores the master replica of the new directory in the same clearinghouse that stores the master replica of the new directory's parent directory.

For example, to create a directory named **/.:/sales** and store the master replica of the new directory in the root directory's initial clearinghouse, enter:

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

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. For example, to place the new directory created in the previous example into another clearinghouse (**/.:/Chicago1_CH**), you would enter the following command:

```
dcecp> directory create /.:/sales -clearinghouse /.:/Chicago1_CH
```

(See the *OS/390 DCE: Command Reference* for complete information on arguments and options to the **directory create** command.)

Checking the ACL Entries for a New Directory

After you create a directory, you want to verify that the users and applications for whom the directory was created have the appropriate permissions. To do this, use the **acl show** command on the directory to see the associated ACL entries. For example:

```
dcecp> acl show /.:/sales
{unauthenticated r--t-}
{group subsys/dce/cds-admin rwdtc}
{group subsys/dce/cds-server rwdtc}
{any_other r--t-}
```

(See the *OS/390 DCE: Command Reference* for complete information on the **acl show** command.)

If the required permissions were not inherited from the new directory's parent directory, use the **acl modify** command to create the necessary ACL entries. For example:

```
dcecp> acl modify /.:/sales -add {user cell_admin rwdtcia}
```

(See the *OS/390 DCE: Command Reference* for complete information on the 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 the 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. After you have verified all affected clearinghouses, you will need to perform another skulk of the directory to finally set the **CSA_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.

Creating a Read-Only Replica

From time to time, you want to create read-only replicas of directories. You create read-only replicas of a directory for the following purposes:

- To distribute the information contained in the directory throughout your network, and to make the information more accessible to users and applications at other locations.
- To 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 contained in the directory.
- To preserve a backup of the information 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 problem at one site does not cause both replicas to be unreachable at the same time. See Chapter 24, “Managing Clerks, Servers, and Clearinghouses” on page 207 for more 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.

You create read-only replicas with the **-replica** option of the **directory create** command. You should create replicas in clearinghouses whose users need to access the directory but do not need (or are not permitted) to update its contents.

Before You Create a 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. For example:

```
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}
```

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. Example:

```
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}
```

The **directory show** command with the **-clearinghouse** and **-replica** options displays all the attribute values for the directory and its replica role.

Note: If any read-only replicas in the directory's existing replica set are unavailable, the replication cannot be completed. The usual skulking process completes the replication as soon as all replicas in the directory's replica set become available.

Permissions for Creating Replicas

To create a replica, you need the following permissions:

- Administer permission to the directory that you intend to replicate.
- Write permission to the clearinghouse that stores the new replica.
- 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 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 Part 8, "DCE Security Service" on page 293 for complete information on using the **acl check** command.

Entering the directory create Command

Use the **directory create** command with the **-replica** and **-clearinghouse** options to create a replica of a directory and store it in the clearinghouse that you specify. For example, the following command creates a replica of the **/.:/mfg** directory and stores the replica in a clearinghouse that is named **/.:/Paris1_CH.**

```
dcecp> directory create /.:/mfg -replica -clearinghouse /.:/Paris1_CH
```

Deleting a Read-Only Replica

Sometimes you may need to delete a replica when the information 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 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.

Permissions for Deleting a Replica

To delete a replica, you must have the following permissions:

- Administer permission to the directory whose replica you want to delete
- Write permission to the clearinghouse from which you are deleting the replica.

Entering the directory delete Command

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 (using the **directory delete** command). See Chapter 28, “Restructuring a Namespace” on page 239 for complete information on how to delete a master replica.

Skulking a Directory

The skulk operation is a periodic distribution of recent modifications 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 start 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 made to a directory's contents or replica set when you do not want to wait for the next skulk.
- Skulk directories that store replicas on servers that were inoperative for an extended period and were just brought back on line.

Permissions for Skulking a Directory

To skulk a directory, you must have the following permissions:

- Administer, write, insert or delete permission to the directory
- The server principal for the server system where you enter the **directory synchronize** command needs read, write, and administer permission to the directory 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 Part 8, “DCE Security Service” on page 293 for complete information on the **acl show** command arguments.

Entering the directory synchronize Command

Use the **directory synchronize** command to start 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 run.) 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 be unsuccessful, especially on directories with 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.

An unsuccessful skulk 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 be unsuccessful, you can determine the cause by reviewing the log of CDS events on the server that stores the master replica of the directory. This log can be found under SDSF in the Interactive System Productivity Facility (ISPF). For example, the following command initiates a skulk on the **/.:/admin** directory:

```
dcecp> directory synchronize /.:/admin
```

Synchronizing CDS Server Clocks

After performing a skulk operation on a directory, you may receive the message, **Server clocks are not synchronized**. If so, you should 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 incorrect (any timestamp greater than five minutes in the future from the current time).

If this is the case, you should adjust the system clock to the current time and then enter the following command:

```
dcecp> clearinghouse repair <clearinghouse-name> -timestamps
```

This command will disable the clearinghouse, analyze and repair timestamps in error, checkpoint the clearinghouse to disk, and re-enable the clearinghouse. To use the command, you need write permission to the server on which the clearinghouse resides. Also, you should run 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.

Changing 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**.

Directory Convergence Set to High

If the directory is set to high convergence and an update is made to the directory, the server that stores the master replica immediately attempts to extend 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. Background skulks occur at least once every 12 hours.

Directory Convergence Set to Medium

If the directory is set to medium convergence and an update is made to the directory, the server that stores the master replica immediately attempts to extend 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 set to medium convergence is skulked at least once every 12 hours.

Directory Convergence Set to Low

If the directory is set to low convergence and an update is made to the directory, the server on which the directory resides makes no immediate attempt to extend updates and waits for the next skulk to synchronize the replica set. A directory set to low convergence is skulked at least once every 24 hours.

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 change this value.

Before You Modify a Directory's Convergence

Before you modify a directory's convergence, you want to 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.

Permissions for Modifying a Directory's Convergence

To change a directory's convergence, you must have write permission to the directory.

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** attribute. 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}
```

Chapter 26. Viewing the Structure and Contents of a Namespace

When you need to view the structure and contents of the cell namespace, you can use one of the programs provided by CDS. The **dcecp** and **cdscp** programs display the information through their command line interfaces. This chapter explains how to use these programs to display namespace information.

Listing the Contents of Directories

The **dcecp** program provides a **directory list** command that lets you 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.

To use the **directory list** command, you must have read permission to the CDS names that you want to display.

For a complete listing of a directory's contents, you enter the **directory list** command with the name of the directory or directories whose contents you wish to view. Example:

```
dcecp> directory list /.:/hosts/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>
```

Note: 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 the **directory list** command. For example, you enter the following command to display the names of all the soft links that are stored in the **/.:/hosts/eng** directory:

```
dcecp> directory list  /.:/hosts/eng/ -links
/.../eng_cell.osf.org/hosts/eng/CDS_CTS /.../eng_cell.osf.org/hosts/\
eng/CDS_UTS
```

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.

To use the **show** command, you must have read permission to the name that you want to display.

In the following example, the **show** command displays 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>
```

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.

To use the **show** command, you must have read permission to the CDS name that you want to display.

You are not permitted to use wildcard characters in the simple names of clerks and servers on the **show** command line.

In the following example, the **show** command displays the current values of all attributes that are associated with the local clerk:

```
cdscp> show clerk
```

The returned display is:

```
                        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
```

Preferred Clearinghouse for Viewing the Namespace

In viewing namespace information using the **show** command of the CDS control program, you can direct these queries to a specific instance of a clearinghouse. The **set cdscp preferred clearinghouse** command specifies a clearinghouse (the **preferred** clearinghouse) that will be used to satisfy read requests from CDS control program commands. If you do not specify a preferred clearinghouse, read requests are made by the CDS control program to the CDS Clerk's cache or to any clearinghouse that is available in the cell.

For more information on the **set cdscp preferred clearinghouse** command, see the *OS/390 DCE: Command Reference*.

Note: When you use the **set cdscp preferred clearinghouse** command, make sure that the name of the clearinghouse parameter is valid. This command does not give an error indication if the specified clearinghouse name does not exist. Also, the **show cdscp preferred clearinghouse** command displays a spurious clearinghouse name that has been previously set using the **set cdscp preferred clearinghouse** command.

Chapter 27. Using the CDS Subtree Commands to Restructure CDS Directories

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**.

Overview of the Merge and Append Procedures

To merge or append CDS directories, you use the DCE control program's (**dcecp**) **directory merge** command. The basic steps for both procedures are as follows:

1. At your system prompt, enter **dcecp** to start 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.

The example merge and append operations described in this section are based on an example namespace, shown in Figure 37.

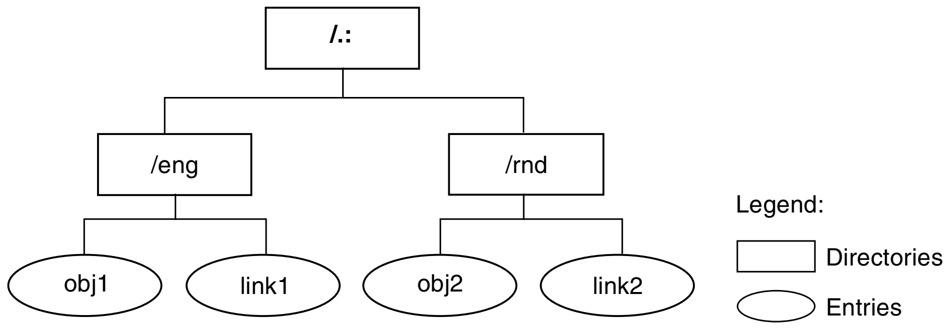


Figure 37. Example Namespace Hierarchy

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**.

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 be unsuccessful.

   ```
   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. If you wish to place child directories in different alternate clearinghouses, you must enter separate **directory merge** commands for each clearinghouse, or you must enter 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 “Handling Errors” on page 235 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 be unsuccessful.

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 run 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.

Figure 38 shows the structure of the example namespace before and after the merge operation in our example.

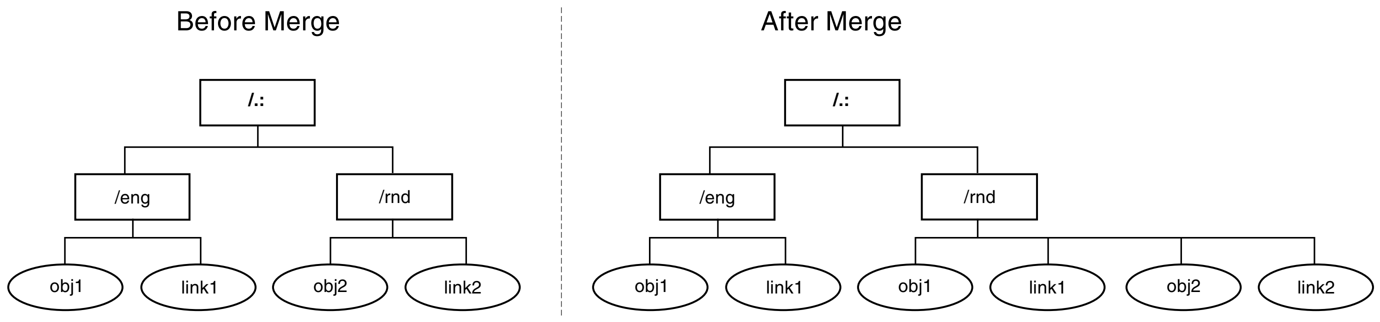


Figure 38. Example Namespace Before and After the Merge Operation

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. If you wish to create a directory in an 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 be unsuccessful:

   ```
   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 any child directories 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 wish 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 be unsuccessful.

   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
   ```

   Figure 39 shows the structure of our example namespace before and after the append operation.

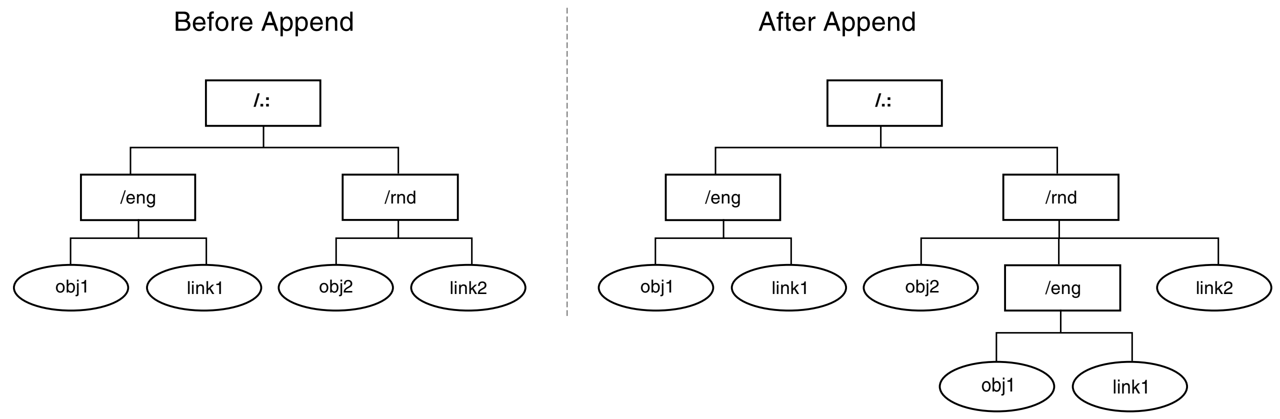


Figure 39. Example Namespace Before and After the Append Operation

Modifying ACLs at the Target Location

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.

Handling Errors

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 because of insufficient permissions

The following subsections explain how to recover from these errors.

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. After 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.

Insufficient Permissions

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 11 on page 236 shows the permissions required to create directories and other CDS object entries at the target.

Table 11. 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.

Merging CDS Directories into a Foreign Cell

You can also 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.

Establishing Cross-Cell Authentication

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 DCE Security part in the book for complete information on how to set up cross-cell authentication.

Performing a Merge Operation into a Foreign Cell

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.

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.

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 after the time of the original merge operation.

Chapter 28. 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 change 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 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 work on the system, or to prepare for removing the system from your network.

This chapter explains how to restructure the namespace.

Note: Managing the clearinghouse must be performed on the host where the clearinghouse resides.

Managing Soft Links

A soft link is an alternative name, or alias, you can use to 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 alternative 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.

Creating a Soft Link

Use the DCE control program's (**dcecp**) **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 (the existing name to which the new soft link points) with the **-linkto** option. You can specify any name in the local cell namespace (or in any named foreign cell namespace) as the destination name, including another soft link.

To create a soft link, you must have insert permission to the directory in which you intend to create the soft link.

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.

Setting Expiration and Extension Values for a Soft Link

All soft links that you create with the **link create** command are permanent and never expire unless you use the command's **-timeout** option to specify an expiration date and time value for 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 = (1996-08-25-16:00:00)**, indicates that if the soft link still exists (has not been deleted manually) on August 25, 1996, at 4 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 for a soft link's **CDS_LinkTimeout** attribute, you can also you can also specify an extension value: 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 again checks (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.

The following command example creates a permanent soft link named **/.:/sales/asia** that points to a directory named **/.:/sales/eur**:

```
dcecp> link create /.:/sales/asia -linkto /.:/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 -linkto/.:/mfg/robotics_controller01 \
>-timeout 1996-12-12-09:00:00
```

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, 1996, at 9 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 -linkto /.:/sales/discount_stats -timeout \
>{1996-01-11-12:00:00 090-00:00:00}
```

In the preceding command, the expiration time placed in the **CDS_LinkTimeout** attribute value indicates that CDS will check that the destination name **/.:/sales/discount_stats** still exists on the next skulk after January 11, 1996, at 12 noon. 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.

Changing a Soft Link Destination Name

Use the **dcecp** program's **link modify** command to specify a new value for a soft link's **CDS_LinkTarget** attribute and redirect the soft link from its current destination name to some other name in the namespace.

To change a soft link's destination name, you must have write permission to the soft link.

For example, 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}
```

Changing a Soft Link Expiration or Extension Value

Use the **dcecp** program's **link modify** command to specify a new value for the expiration and extension values stored in a soft link's **CDS_LinkTimeout** attribute. Even if you want to change only one of the values, you must specify values for both expiration and extension in your command. You specify a new value in the same format 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*.

To change a soft link's expiration or extension value, you must have write permission to the soft link.

The following command example sets the expiration value of a soft link named **/.:/eng/link01** to December 31, 1996, at 12 noon. In this example, no extension is currently assigned to the soft link.

```
dcecp> link modify /.:/eng/link01 -change {CDS_LinkTimeout 1996-12-31-12:00:00 \
>000-00:00:00}
```

The following command sets the expiration value of a soft link named **/.:/eng/link01** to December 31, 1996, at 12 noon and sets the soft link's extension value to 90 days:

```
dcecp> link modify /.:/eng/link01 -change {CDS_LinkTimeout 1996-12-31-12:00:00 \
> 090-00:00:00}
```

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 a soft link's assigned expiration and extension times have been reached, you can delete the soft link from the namespace yourself.

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.

For example, the following command deletes a soft link named **/.:/dist/pointer_1**:

```
dcecp> link delete /.:/dist/pointer_1
```

Deleting Entries Pointed to by Soft Links: The **delete link** subcommand does not delete the object pointed to by the soft link. To delete the object pointed to by the soft link, use the **delete object** subcommand. For example, if **/.:/dist/pointer_1** is linked to **/.:/temp/object_1**, running the command:

```
cdscp> delete object /.:/dist/pointer_1
```

deletes **/.:/temp/object_1**.

Showing Entries Pointed to by Soft Links: The **show object** subcommand shows information about an object pointed to by a soft link. In the previous example, the command:

```
cdscp> show object /.:/dist/pointer_1
```

displays information about **/.:/temp/object_1**.

Changing a Directory Replica Set

A directory's replica set always contains a master replica; it can also contain other read-only replicas. The values stored in the **CDS_Replicas** attribute 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 CDS control program's (**cdscp**) **set directory to new epoch** command to overwrite the current values 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.

Before You Change a Directory's Replica Set

Before you change 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 you use to change 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 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 of a 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 Chapter 26, “Viewing the Structure and Contents of a Namespace” on page 227 complete information on how to use the **dcecp** program's **directory show** command.

Permissions Required for Modifying a Replica Set

To change a directory's replica set, you must have the following permissions:

- Administer permission to the directory. Also, the server principal 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 permission to the directory.

When you know which replicas to include and exclude and have changed permissions that need to be changed, enter the **cdscp set directory to new epoch** command to modify a directory's replica set. Instructions for your two options (1) designating a new master replica, and (2) excluding an existing read-only replica are given in the sections that follow.

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 whose clearinghouse contains one or more master replicas will be down for an extended period of time or removed permanently from the network
- A clearinghouse that stores one or more master replicas will be deleted from the namespace
- You want to locate a master replica closer to where the majority of updates to the directory originate.

To designate a new master replica, use the **cdscp** program's **set directory to new epoch** command.

Figure 40 on page 243 illustrates an example replica set. This 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**).

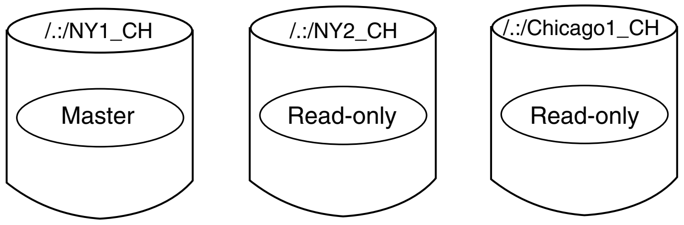


Figure 40. Example Replica Set

The following command designates the read-only replica 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 read-only /.:/NY1_CH \
> /.:/NY2_CH
```

Figure 41 shows the result of the preceding command.

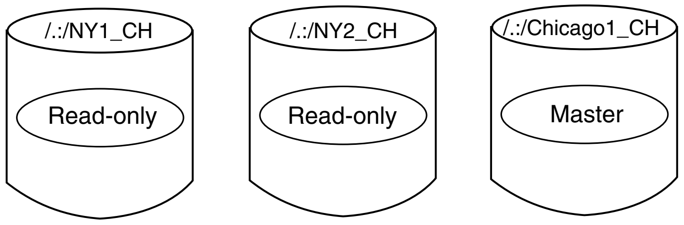


Figure 41. Example Replica Set After Master Redesignation

Excluding a Replica from a Replica Set

You can temporarily exclude a replica from its replica set when the clearinghouse in which the replica is stored unexpectedly becomes unavailable. This makes it possible for CDS to complete skulks of the directory during the time the excluded replica is unavailable.

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.

In this example, the replica set of the **/.:/eng** directory consists of three replicas: the master replica, stored in clearinghouse **/.:/Chicago1_CH**, and 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 not succeed unless you temporarily exclude the read-only replica 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 stored in the **/.:/NY2_CH** and **/.:/Chicago1_CH** clearinghouses:

```
cdscp> set directory /.:/eng to new epoch master \
> /.:/Chicago1_CH read-only /.:/NY2_CH \
> exclude /.:/NY1_CH
```

Figure 42 shows the result of the preceding command.

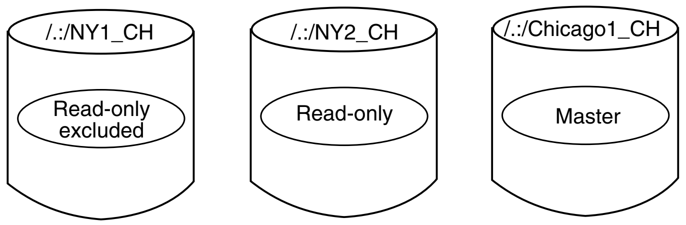


Figure 42. Example Replica Set After Replica Exclusion

When connectivity with the **/.:/NY1_CH** clearinghouse is reestablished, enter the following command to reintroduce the read-only replica stored in clearinghouse **/.:/NY1_CH** to the replica set:

```
cdscp> set directory /.:/eng to new epoch master \
> /.:/Chicago1_CH read-only /.:/NY1_CH \
> /.:/NY2_CH
```

Note: Always reintroduce excluded replicas to their replica sets as soon as possible after the clearinghouse in which they reside again becomes available.

Deleting Directories

You may sometimes want to delete a directory from your namespace when the users no longer need the information it contains. 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 considerations are discussed in following sections.

To delete a directory, you must have the following permissions:

- Delete permission to the directory
- Write permission to the clearinghouse that stores the master replica of the directory
- The server principal for the server from which you enter the **directory delete** command needs administer permission to the parent directory or delete permission to the child pointer that points to the directory you intend to delete.

  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 of the **dcecp** utility and verify that the server principal has the appropriate permissions. See Part 8, "DCE Security Service" on page 293 for complete information on using the **acl show** command.

Deleting a Non-Replicated Directory

To delete a directory that has no replicas, use the **dcecp** utility's **directory delete** command. 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 confirm that you want to delete the directory that you specify in 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 Chapter 8, " Writing Scripts and dcecp Objects" on page 69 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.

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
```

If a directory to be deleted is not empty, the **directory delete** command will not succeed. To recover from this kind of problem, 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** program **directory show** command. Look at the **CDS_Replicas** attribute of the directory in the list. Each replica's **CDS_Replicas** attribute has several sub-attributes. Look at the **CH_Name** sub-attribute for each replica to get the name of the clearinghouse where it is located. For example:

```
dcecp> directory show /.:/sales
{RPC_ClassVersion {01 00}}
{CDS_CTS 1996-05-06-11:41:05.314-05:00I0.000/08-00-09-25-13-52}
{CDS_UTS 1996-06-21-03:06:08.842-05:00I0.000/08-00-09-25-13-52}
{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 1996-08-24-19:30:30.827}
   {extension +1-00:00:00.000I0.000}}
  {myname /.../absolut_cell/sales}}}
{CDS_DirectoryVersion 3.0}
{CDS_ReplicaState on}
{CDS_ReplicaType Master}
{CDS_LastSkulk 1996-01-29-19:45:44.841-05:00I0.000/08-00-09-25-13-52}
{CDS_LastUpdate 1996-06-21-03:06:08.842-05:00I0.000/08-00-09-25-13-52}
{CDS_Epoch 60ac0730-bf9b-11cd-9362-080009251352}
{CDS_ReplicaVersion 3.0}
```

The 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. An example command line is the following:

```
dcecp> directory delete /.:/sales -replica -clearinghouse /.:/NY1_CH
```

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 Chapter 8, " Writing Scripts and dcecp Objects" on page 69 for a discussion of writing scripts.

Relocating a Clearinghouse

Note: This section describes the procedure that you use to temporarily relocate a clearinghouse from one CDS server system to another. Note that the procedure cannot be used to configure additional CDS server systems. (See *OS/390 DCE: Configuring and Getting Started* 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.

- 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.

To relocate a clearinghouse, follow these steps:

1. Dissociate 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.

Dissociating 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.

To use the **clearinghouse disable** command, you must have write permission to the server on which the clearinghouse resides.

The following example command removes knowledge of clearinghouse **/.:/Chicago2_CH** from the memory of its host server:

```
dcecp> clearinghouse disable /.:/Chicago2_CH
```

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 *nnnnnnnn* represents an 8-digit number.

You should verify the existence of these files before you attempt to copy them to the new host system. (See the *OS/390 Program Directory* for the full pathnames 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 malfunction or other interruption during the clearinghouse's most recent checkpoint operation. If you do find two files, copy both of them to the target server system. The server software that is 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**.

The **.checkpoint***nnnnnnnn* and **.tlog***nnnnnnnn* files must be moved in binary format. The **ftp** utility lets you move files in binary format. The **.version** file must be moved in regular character format. The **ftp** utility also lets you do this.

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, then starts the clearinghouse.

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.

In the preceding example, the database files for clearinghouse **/.:/Chicago2_CH** were successfully copied to a server system named **orion**. The following command, which is issued on **orion**, relocates the clearinghouse named **/.:/Chicago2_CH** on that server:

```
dcecp> clearinghouse create /.:/Chicago2_CH
```

To test that the move succeeded, enter a **dcecp clearinghouse show** command for the newly relocated clearinghouse. Verify that the clearinghouse now resides on the target server system by looking at the tower address.

```
dcecp> clearinghouse show /.:/Chicago2_CH
```

If the clearinghouse is not found, or is found to be at the wrong tower address, try one of the methods that follow. Try the first method, and if that does not work, try the second. The second method works only on OS/390.

1. Skulk the root directory, then try to show the clearinghouse again:

   ```
   dcecp> directory synchronize /.:/
   dcecp> clearinghouse show /.:/Chicago2_CH
   ```

   If the clearinghouse is still not found or is at the wrong tower address or you cannot skulk the root directory, follow the steps associated with recovering from a corrupted CDS cache. This is most likely to occur if the source server is also the primary CDS server. The recovery should take place on the primary CDS server. Because each platform (for example AIX or OS/390) might have a different set of instructions for recovering from a corrupted CDS cache, it is advisable to refer to the documentation for that platform. The steps for OS/390 are in “Recovering from a Corrupted CDS Cache” on page 211.

2. On the primary CDS server, stop the Advertiser, Clerk, and CDS daemons and generate a new cache.

   a. Enter these commands from the operator console:

   ```
   modify dcekern, stop cdsadv
   modify dcekern, stop cdsclerk
   modify dcekern, stop cdsd
   ```

   b. Enter these commands from **hfs**:

   ```
   rm /opt/dcelocal/var/adm/directory/cds/cds_cache.version
   rm /opt/dcelocal/var/adm/directory/cds/cds_cache.nnnnnnnnnn
   ```

where *nnnnnnnnnn* is a 10-digit number.

c. Enter these commands from the operator console:

```
modify dcekern, start cdsadv
modify dcekern, start cdsclerk
modify dcekern, start cdsd
```

d. Enter these commands from **hfs**:

```
dcecp cdscache create cache_name -binding server_binding
```

where *cache_name* is a simple name for the cached server and *server_binding* is the protocol sequence and network address of the server node. The string format is *protocol-sequence:network-address*.

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.

Before You Delete a Clearinghouse

Before you attempt to delete a clearinghouse, make sure of 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 subsequently try to delete the clearinghouse, CDS will not find it and will return 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 “Changing a Directory Replica Set” on page 241 for instructions on modifying a directory's replica set.)

Permissions for 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.

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.

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 example command deletes the **/.:/Paris2_CH** clearinghouse:

```
dcecp> clearinghouse delete /.:/Paris2_CH
```

Chapter 29. 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 communication, serving as a connection to other cells through the global naming environment. This chapter describes how the GDA works in facilitating intercell communications.

The chapter also describes how to define the local cell in one of the global naming environments (GDS, DNS, or LDAP), a step that is necessary to make the local cell accessible to other cells. DCE does not support cells registered simultaneously in GDS and DNS.

Note: If you are going to perform intercell communications, make sure that the **BIND_PE_SITE** environment variable in your environment variable file is set to 0. You can set this by editing the **envar** file in your home directory.

How the Global Directory Agent Works

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, DNS, or LDAP namespace and returns the results to the clerk. The clerk then uses those results to contact a CDS server in the foreign cell.

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 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 43 on page 252 shows how a CDS clerk and CDS server interact to find a GDA.

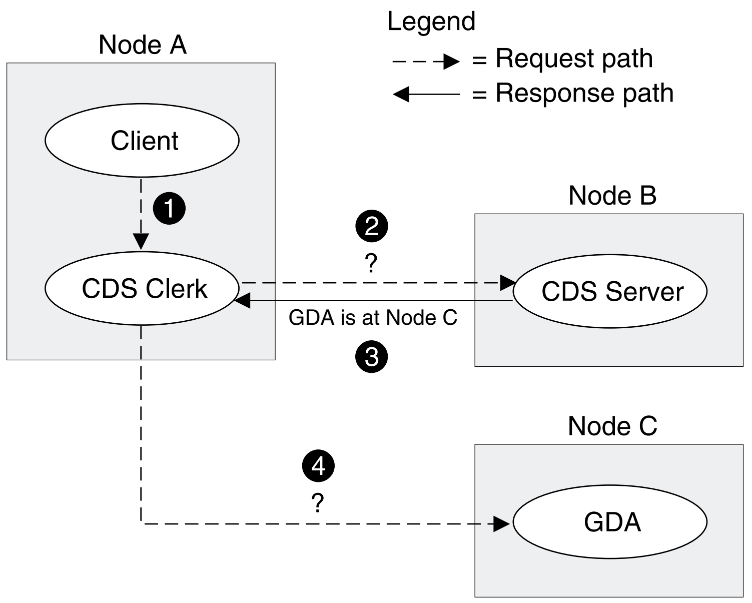


Figure 43. How the CDS Clerk Finds a GDA

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 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 44 on page 253 shows how CDS and a GDA interact to find a name in a foreign cell defined in DNS. Suppose the name is **/.../widget.com/printsrv1**, which represents a print server in the foreign cell.

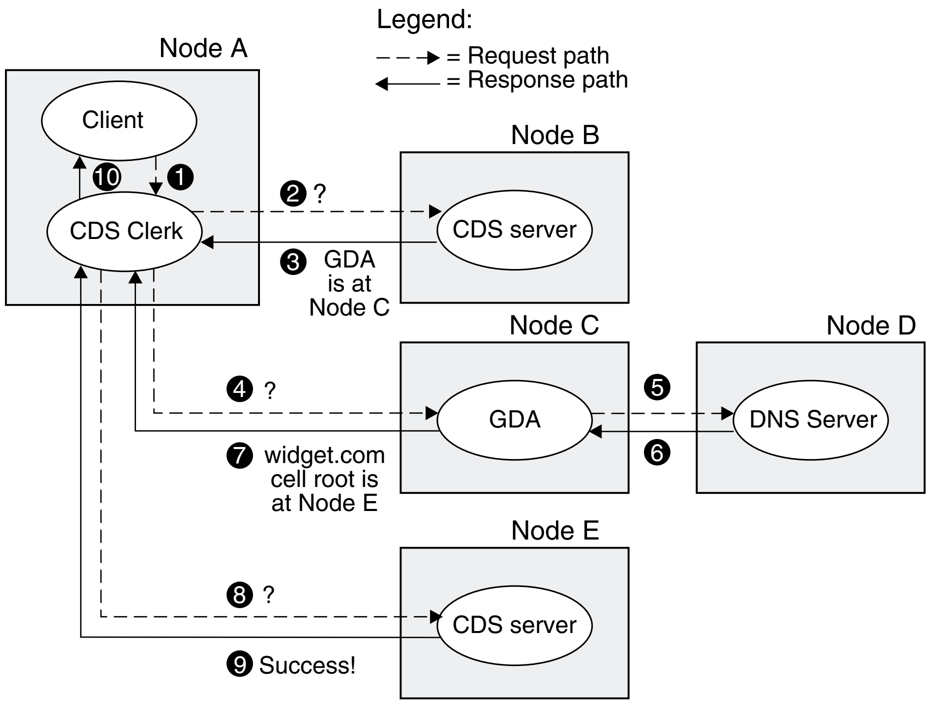


Figure 44. How the GDA Helps CDS Find a Name

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 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 DNS-style, so it assumes the second component is a cell name 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 running of operations is nearly identical for an X.500 name. If the GDA recognizes that a name is a typed name, it passes the name to an LDAP server, rather than to a DNS server.

6 DNS looks up and returns to the GDA information 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, so 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 might actually receive referrals to several servers in the foreign cell.

10 The clerk passes the information to the client application that requested it.

Note that both of the previous examples represent initial lookups. The CDS clerk caches the locations of GDAs after 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.

Note also that a GDA knows its own cell name and can therefore avoid contacting a global directory service to look up names in its own cell. Further, the GDA can recognize whether a cell name conforms to X.500 or DNS naming syntax, and it uses that knowledge to route a lookup request to the appropriate global directory service.

Managing the Global Directory Agent

Configure the GDA using the DCE configuration program; the GDA requires little management after it is configured. (See the *OS/390 DCE: Planning* for details on configuring the GDA.)

The GDA is typically started and stopped automatically by scripts that run as part of usual 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**. To start the **gdad** process, follow these steps:

1. Make sure that a CDS server and the CDS Clerk are 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:

   ```
   MODIFY DCEKERN, query all
   ```

   If the **dced** process appears on the list of active processes, proceed to Step 5. If the **dced** process does not appear on the list of active processes, enter the following command to start the process:

   ```
   MODIFY DCEKERN, start dced
   ```

4. Enter the following command to start the **cdsadv** process:

   ```
   MODIFY DCEKERN, start cdsadv
   ```

5. Enter the following command to start the **gdad** process:

   ```
   MODIFY DCEKERN, start gdad
   ```

To stop the GDA, enter the following command:

```
MODIFY DCEKERN, stop gdad
```

Enabling Other Cells to Find Your Cell

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 (DNS or LDAP server). Before you do this, obtain a unique cell name from the appropriate naming authority. For details, see *OS/390 DCE: Planning*.

After you configure a cell, name it, and initialize the cell namespace, you can use the CDS control program (**cdscp**) **show cell** command to obtain data that you need to create or change the cell entry in DNS. 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. This section describes how to define and maintain a cell entry in DNS or LDAP server. It assumes a basic familiarity with DNS and LDAP server; for details, see the appropriate documentation for each global name service.

Defining a Cell in the Domain Name Service

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 hostname of the system where the CDS server resides. 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

The following example shows a set of resource records for a cell named **dcecell14.endicott.ibm.com**, in which one replica of the root directory exists. 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 IN class indicates that the protocol is Internet.

```
;BEGIN DCE CELL /.../dcecell14.endicott.ibm.com INFORMATION
;Primary CDS server
dcecell14.endicott.ibm.com.          IN     MX     1      dcehost.
endicott.ibm.com.
dcecell14.endicott.ibm.com.          IN     A      9.130.44.50
dcecell14.endicott.ibm.com.          IN     TXT    "1 3edbc070-6974-
11d0-9b90-08005a191a6c Master /.../dcecell14.endicott.ibm.com/dcecell14
_ch dcehost.endicott.ibm.com"
;END DCE CELL /.../dcecell14.endicott.ibm.com INFORMATION
```

After you configure a cell, you can use the CDS control program (**cdscp**) **show cell** command to display the information 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 **dcecell14.endicott.ibm.com**. Note that the **show cell** command does not display the host name (including domain) at the end of the TXT record.

```
cdscp> show cell /.../dcecell14.endicott.ibm.com as dns

                        SHOW
                        CELL   /.../dcecell14.endicott.ibm.com
                          AT   1997-03-03-12:20:56
                         TXT = 1 3edbc070-6974-11d0-9b90-08005a191a6c Master
/.../dcecell14.endicott.ibm.com/dcecell14_ch 3e5a0e7c-6974-11d0-9b90-08005a191a6c
```

To create new resource records in the DNS space, run the **mkdceregister** utility. This utility generates the data automatically for you. You can run **mkdceregister** from the DCE configuration program panels or from a shell script in the HFS. Take the output from **mkdceregister** and add it to the appropriate DNS data file.

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.

Note: See *OS/390 DCE: Planning* for complete information on how to contact the NIC Domain Registrar to register a domain name.

Defining a Cell in an LDAP Server

Note: The LDAP server is currently not available on OS/390 DCE.

In LDAP, cell information is contained in two attributes: **CDS-Cell** and **CDS-Replica**. You can cause an existing LDAP name to become a cell entry by adding these two attributes to the name. If the name you want to use for the cell does not yet exist, you must create it and then add the attributes.

The **ldap_addcell** utility automatically generates the **CDS-Cell** and **CDS-Replica** attribute information and then adds it to the LDAP server. This utility can be run from the DCE configuration panels, the UNIX System Services HFS environment, or from the batch environment. See the *OS/390 DCE: Command Reference* or the *OS/390 DCE: Configuring and Getting Started* for more information.

Part 7. DCE Distributed Time Service

Chapter 30. Introduction to the DCE Distributed Time Service 259
Distributed Time Service Advantages 260
  Applications Support 260
  External Time Provider Support 260
  Manageability 261
  Quantitative Inaccuracy Measurement . . 261
Basic Distributed Time Service Concepts . . 262
  Time Measurement Factors 262
  Inaccuracy Values 262
  Synchronizing System Clocks 263
  How the Distributed Time Service Adjusts System Clocks 265
  Distributed Time Service Time Representation 265
How DTS Works 268
  Clerks 268
  Servers 269

Chapter 31. Managing the Distributed Time Service 271
Using the dcecp Program 271
  DTS Objects 271
  DTS Command Operations 271
  DTS Object Attributes and Counters 272
DTS Timestamp Format 273
Reconfiguring DTS on Nodes 274
  Stopping an Existing Clerk or Server . . . 275
  Creating a New Clerk or Server 275
  Setting Clerk and Server Attribute Values .275
Modifying Clerk and Server Attributes 276
  The minservers Attribute 277
  Use of minservers Attribute with Global Servers 278
  Use of minservers Attribute with Systems on Point-to-Point Lines 279
  The maxinaccuracy Attribute 279
  The syncinterval Attribute 280
  The tolerance Attribute 281
  The localtimeout, globaltimeout, and queryattempts Attributes 282
  The serverentry and serverprincipal Attributes 282
Management Tasks Specific to Servers . . . 283
  Designating Global and Courier Servers . 283
  Designating Global Servers Outside a Cell 284
  Matching Server Epochs 285
  Setting the checkinterval Attribute for Connection to a Time Provider 285
Changing the System Time 286
  Updating the Time Monotonically 286
  Updating the Time Nonmonotonically . . . 286
  Forcing System Synchronization 287
Controlling Access to DTS 287

Chapter 32. OS/390 DCE Considerations in DTS 289
DCE Software Clock 289
Null Time Provider 289
  What Is the Null Time Provider Program? .289
  DTS and the OS/390 Sysplex Environment 290
Setting the Time Zone 290
Overriding the System Time Zone 291
  Using POSIX Time Format in TZ 291
Resetting the DCE Software Clock 291

Chapter 30. Introduction to the DCE Distributed Time Service

This chapter gives a conceptual overview of the DCE Distributed Time Service (DTS) and its functions. Some basic time and clock concepts, DTS time representation, and basic DTS operation are also presented.

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 provided by DTS enables distributed computing applications to determine duration, and perform event sequencing 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 DTS, a client system or application obtains the time from the clock that is maintained by the **DTS entity**. In OS/390 DCE, this is called the **DCE software clock**. The DTS entity can be a **DTS server**, or a **DTS clerk**. DTS clerks and servers also obtain time from DTS servers on other hosts. (Note that throughout this document, the term **entity** refers to the server or clerk process.)

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. A DCE node is either a server or a clerk. 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 the network.

Because no device can measure the exact time at a particular instant, DTS expresses the time as an **interval** containing 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 non-intersecting 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 disseminate them in the network. The TPI also permits other distributed time services to interoperate with DTS.

DTS provides many other valuable services for computer networks running distributed applications. A summary of its major features and benefits follows:

- **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 reported by entities.
- **Fault Tolerance**. DTS reports faulty servers and does not use their time values during clock synchronizations.
- **Management Capability**. The DTS control programs (**dcecp** and **dtscp**) enable you to control and monitor the DTS entity.

- **Application Programming Interface (API)**. DTS provides an interface that allows applications to obtain, compare, and 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 usually provides unidirectional clock adjustment. You can use the DCE or DTS control program, though, for non-monotonic clock adjustment if desired.
- **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.

Distributed Time Service Advantages

DTS offers all the features usually provided by a time service, but it also has several features that enhance network performance.

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 (this applies to 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 refer to 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 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 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 *OS/390 DCE: 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 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.

Note: External time provider programs are not available in OS/390 DCE. However, the interface to communicate with a user-written time provider program is available. OS/390 DCE provides the null time provider program.

DTS servers can synchronize with time providers by means of the Time Provider Interface, which is described in *OS/390 DCE: 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 disseminated throughout the network. See “Basic Distributed Time Service Concepts” on page 262 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 administrators to synchronize system clocks after DTS is initially configured. DTS can be fault tolerant provided that an adequate number of servers exist. 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 because of varying network conditions
- Modify the role of the DTS entity when the network topology changes (for example, from clerk to server, server to clerk).

DTS provides a full-featured management interface that you use to adjust system clocks, change the values of the DTS management parameters, monitor synchronization information, 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 administrators 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” on page 263 for further information about the DTS synchronization process.

Basic Distributed Time Service Concepts

This section 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 progressing to the chapter on managing the DCE Distributed Time Service.

Time Measurement Factors

This section 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 tends 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 unsynchronized.

The difference between any two clock readings is known as the **skew** between the clocks. The clocks used in many computer systems have a manufacturer-specified drift. 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 system 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: Communication 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 required to send and receive messages between systems. Because of the varying quality of communications links, however, the time 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 communications links.

Inaccuracy Values

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. DTS uses an inaccuracy value 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 three factors:

Drift. When reading a clock, DTS calculates the maximum amount of time that the clock may have drifted after 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. DTS 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 the same 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. Because of the slowing of the earth's rotation, it occasionally becomes necessary to advance UTC time by one second; these events are known as **leap seconds**. Leap seconds may occur in the final second of any month, and usually occur about once every 18 months. At the end of each month, DTS accounts for leap seconds by increasing all inaccuracy measurements by one 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), has zero inaccuracy, and a system clock drift of 8 seconds per day. 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 is probably somewhere in between. The system time is an interval containing UTC time and bounded by the inaccuracy, as shown in Figure 45. Using the DTS format for displaying time, the combined time and inaccuracy interval is expressed as follows: 1990-08-03-00:00:00.000I08.000.

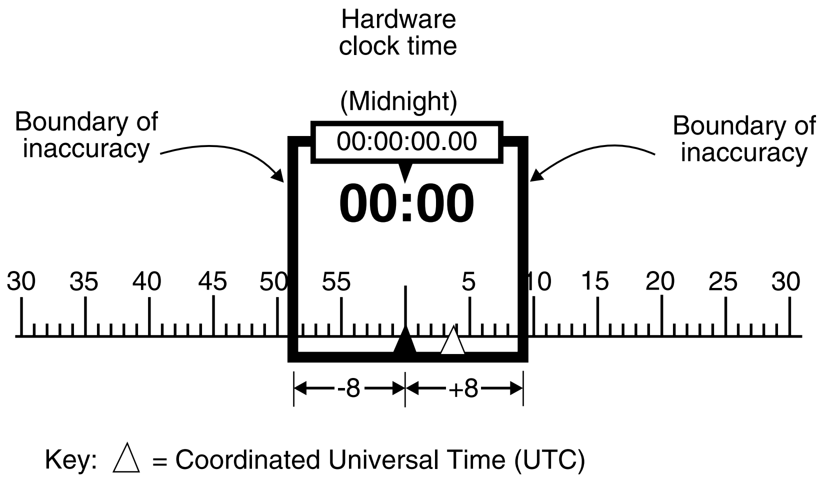


Figure 45. Time and Inaccuracy

Synchronizing System Clocks

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

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 entity requests time intervals from several servers, it uses them to calculate a new time that is correct (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 entity has a set of intervals (t1 through t4 in Figure 46 on page 264). Because each interval contains UTC, the intersection is the smallest interval the entity 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 received from faulty clocks (t2 in Figure 46). DTS detects and rejects clock intervals that do not intersect with the majority of the intervals. When DTS detects a faulty interval, it displays 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 46), but the large time interval is ignored because more accurate times are available. Note that in Figure 46, endpoints of correct time t1 are further from the computed time midpoint than those of the interval that is declared faulty (t2).

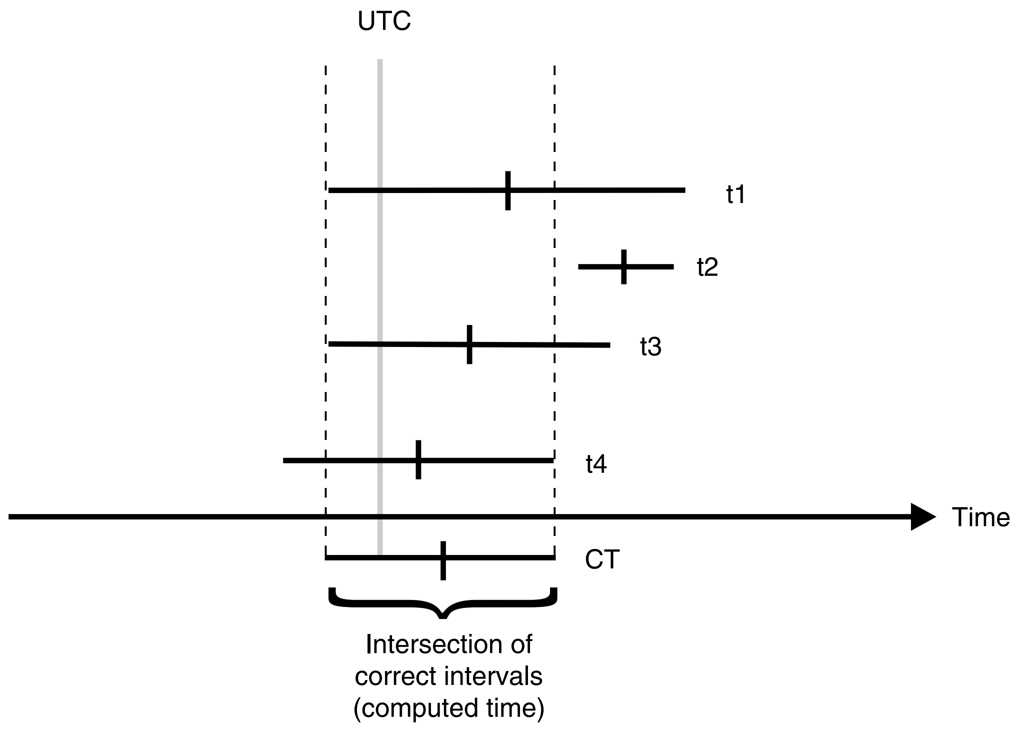


Figure 46. Computed Time

During the synchronization process, servers with the greatest accuracy have the most influence in determining new system times throughout the network. In Figure 46, 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 supplied by any single clock. Note that the computed time in Figure 46 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 the Distributed Time Service Adjusts System Clocks

DTS adjusts system clocks at the rate of 100 to 1; 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 usual rate of drift, so that synchronization is carried out without further significant interference from the clock.

Figure 47 illustrates how DTS changes the increment to the software clock. The top line represents a 10-millisecond increment to the usual clock at every 10-millisecond tick. The middle line illustrates the adjustment to a fast clock; DTS slows the clock by incrementing the clock by 9.9 milliseconds instead of 10 at each tick. The bottom line illustrates the adjustment to a slow clock; DTS speeds it up by incrementing the clock by 10.1 milliseconds instead of the usual 10 at each tick.

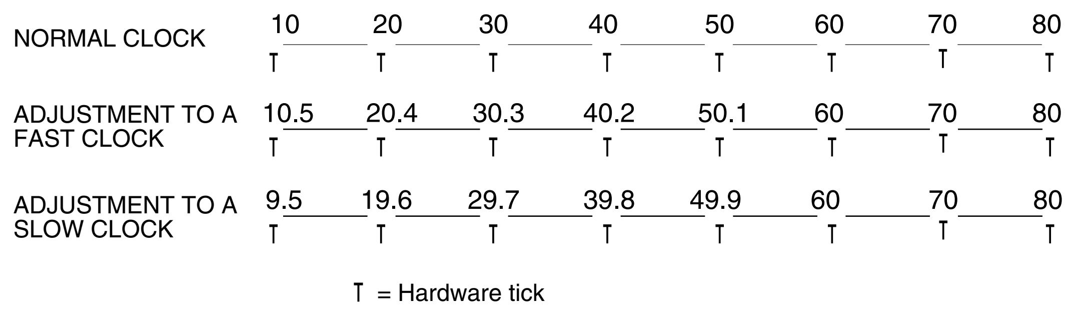


Figure 47. Adjustment of the Clock

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 serious hardware problems, causing a large number of the clocks to become faulty
- When the time interval for a given clock does not intersect with the intervals of other clocks, and the error exceeds a predetermined tolerance.

Distributed Time Service 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 representing UTC. You cannot read or disassemble a DTS binary timestamp; the DTS Application Programming Interface (API) allows other applications to convert or manipulate the timestamps. To display the timestamps, DTS translates them into printable text.

Absolute Time: An **absolute time** is a point on a time scale. For DTS, absolute times refer to 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 DCE control program **dcecp clock show** command to display an absolute time, it is converted to printable text string as shown in the following display:

```
1990-11-21-13:30:25.785-04:00I010.082
```

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

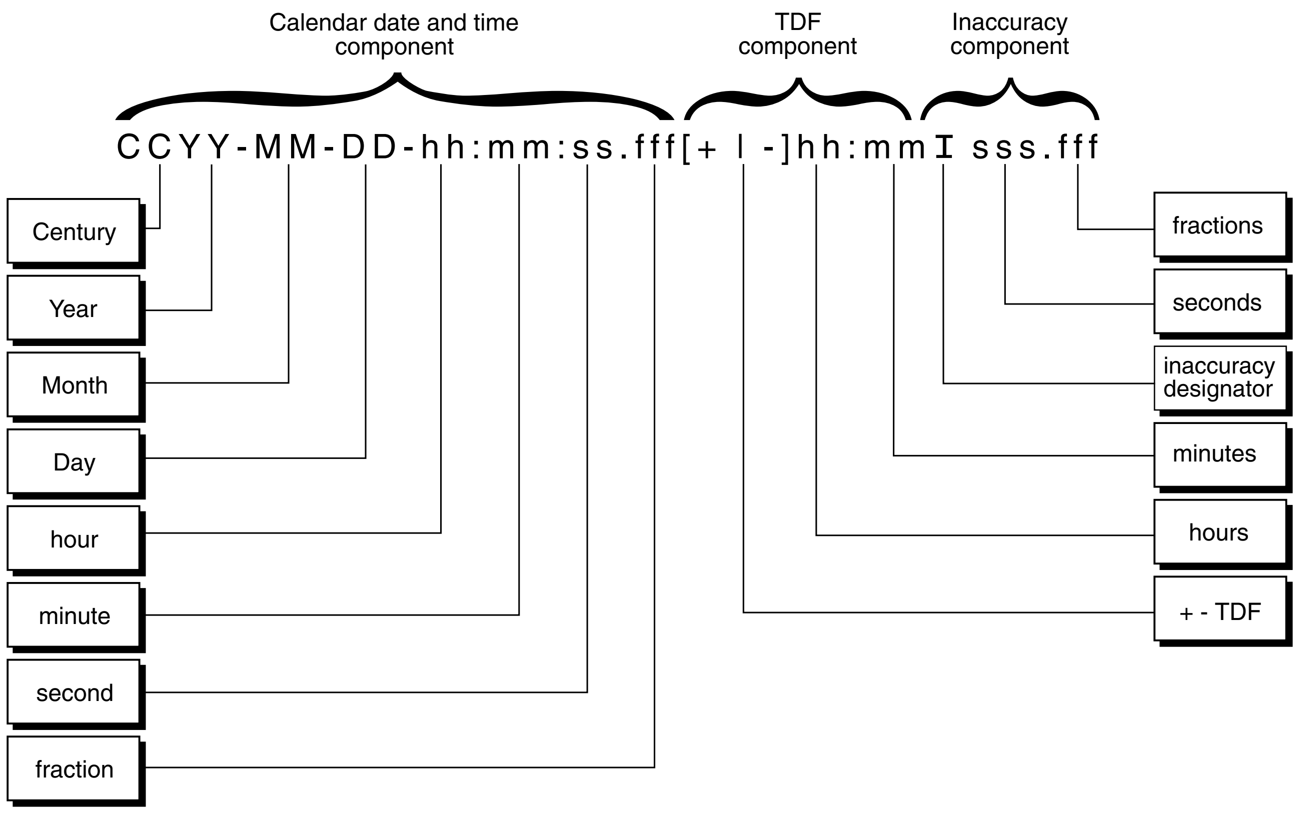


Figure 48. ISO-Compliant Time Format

In this format example, the relative time preceded by the + (plus sign) or - (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 associated with the time.

You can express the DTS time you want to display in several ways; the DTS time BNF format is defined in Appendix F, "DTS Extended BNF" on page 539.

Although the DCE control program **clock show** command displays all times in the previous format, the interface also accepts these variations to the ISO format on input:

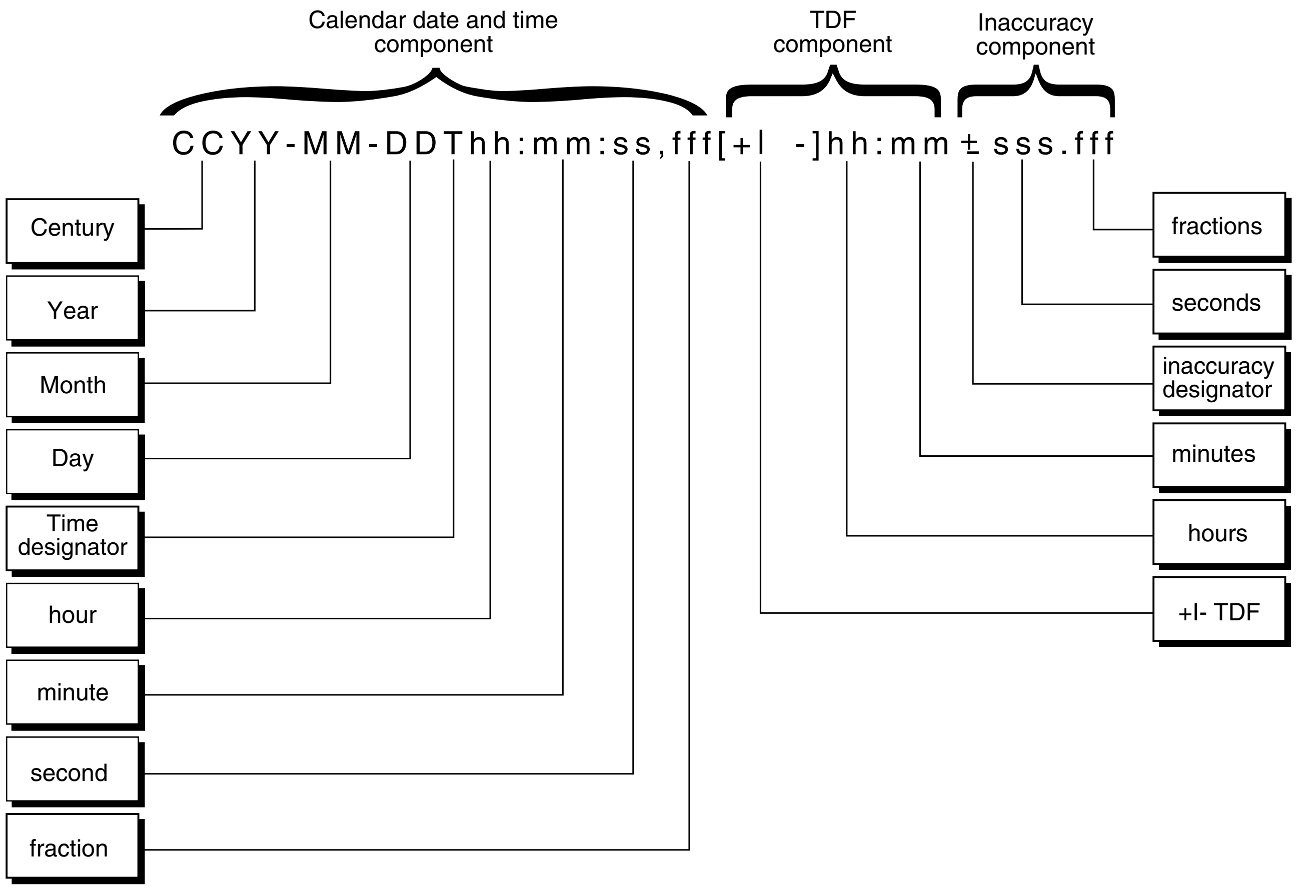


Figure 49. ISO-Compliant Time Format Variation

In this format, the delineator T separates the calendar date from the time, a 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 ([+I–]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.

The following section describes relative time, which is derived from absolute time.

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

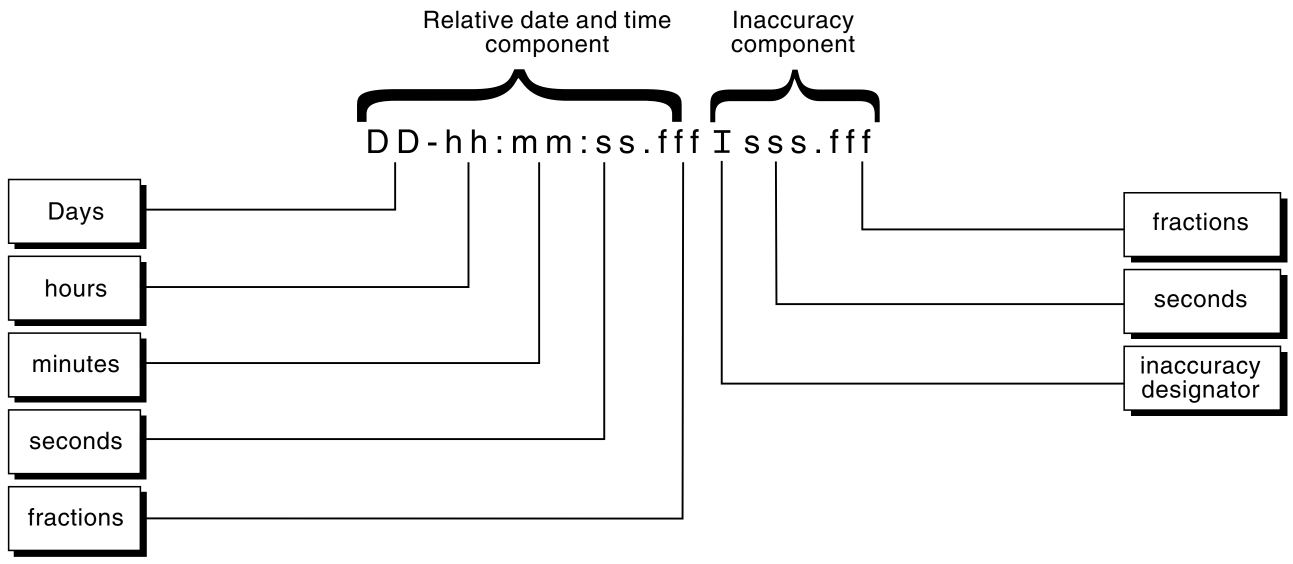


Figure 50. Relative Time Format

The simple relative times that you specify with DTS-related **dcecp** commands use neither the calendar date nor inaccuracy fields, because these fields are associated with absolute times. Positive relative times are not signed, while negative relative times are preceded with - (minus sign). 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, 1995-11-30-00:30:25.000I00.030 minus 00-00:15:25.000 equals 1995-11-30-00:15:00.000I00.030.

How DTS Works

DTS has two major software components: clerks and servers. The following sections describe each of these components and tell how they interact to provide time to client applications and to synchronize system clocks.

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.

To build server lists and synchronize with the servers on the list, clerks need to be able to locate servers automatically. Clerks discover servers 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. If the profiles have not been configured in any other way, the clerk searches for the DTS servers in the following sequence. Initially, the clerk looks at the machine profile of the node where it resides (/.:/hosts/*hostname*/profile). This profile has a default entry, which points to the cell profile (/.:/cell-profile). In the cell profile, the clerk looks for Time Server interfaces that are available for synchronization. There is a **LAN Server Interface** in the cell profile which points to the **LAN profile**. The LAN profile will contain some local time server entries which will be used by the clerk to build up its list of time servers.

Note that when a Time Server entry exists in the Cell profile, it is also a Global Time Server. If the entry exists in the LAN profile it is a Local Time Server.

If a clerk does not obtain enough server entries as dictated by the DTS management attribute **minservers**, it attempts to locate additional servers, usually those outside the **local set**. 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 usually found outside the **local set**. See the following section for further information on servers.

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 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 spread the precise time intervals they obtain from the time providers throughout the network.

Epochs divide the DTS implementation into logically separate areas. Servers obtain time values from other servers, but only synchronize with those servers that have the same **epoch number**. All servers have the same epoch number when they are created. Infrequently, you may want to change a server's epoch number (using the management interface) to isolate it from the network in order to correct a problem.

The rest of this section describes the three subtypes of DTS servers.

The Local Server Set: A set of **local servers** reside on the same network 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 usually 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 shorter intervals, clerks request time values from the local servers. Clerks and servers, however, need only 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 in a single LAN, but **global servers** are available to all LANs within a cell. Any server can be configured as either a local or a global server. (See the **dts configure** command in the *OS/390 DCE: 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 in the following situations:

- When a network has multiple LANs or an extended LAN
- When systems that are not on LANs have access to LANs through point-to-point links
- When clerks or local servers cannot access the required number of local servers determined by the **minservers** attribute.

You can reconfigure a local server as a global server by using the **dcecp dts configure** command with the **-global** option. Configuring a server as global 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 mandated by the **minservers** attribute. Certain local servers also regularly request the time from global servers; see the following section.

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 you use the **dcecp dts modify** command with the **-change** option.

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 (depending on the number of time servers) and one global server when synchronizing their own clocks. Couriers provide network-wide synchronization through the following procedure:

1. 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.
2. Couriers use the global server times and local server times to synchronize their clocks with their respective systems.
3. 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 network address regularly synchronizes with global servers until a courier is again available.

If a courier cannot find any global server, it uses local servers and increments its **no global servers detected** count.

Chapter 31. Managing the Distributed Time Service

This chapter describes management tasks that you perform for the DCE Distributed Time Service (DTS). The DCE control program (**dcecp**) has commands that you can use for performing these tasks. The chapter contains brief descriptions of these commands. Detailed descriptions of the commands appear in the *OS/390 DCE: 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 *OS/390 DCE: Command Reference* for detailed descriptions of the **dtscp** commands for managing DTS.

Using the dcecp Program

Because detailed information about **dcecp** and its command syntax appears in Chapter 7, "DCE Control Program Introduction" on page 45 this chapter does not repeat that information. It describes only the commands that the **dcecp** program provides specifically for managing DTS.

The **dcecp** 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 subsections describe the DTS objects that the **dcecp** program operates on and the types of operations that the control program can perform on these objects.

DTS 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.
- 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

Table 12 summarizes the operations performed by the **dcecp** object commands on DTS objects.

Table 12 (Page 1 of 2). DTS Command Operations

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.

Table 12 (Page 2 of 2). DTS Command Operations

Command Operations	Descriptions
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 allowed only to view the values (by using the **dcecp dts show** command). Other attributes contain values that you can reset according to the needs of your environment (by using the **dcecp dts modify** command). Counters are used internally by the DTS daemon and contain values that you can only view.

Table 13 lists the server and clerk attributes that you can set.

Table 13. Settable DTS Object Attributes

Servers	Clerks
checkinterval	—
courierrole	—
epoch	—
globaltimeout	globaltimeout
localtimeout	localtimeout
maxinaccuracy	maxinaccuracy
minservers	minservers
queryattempts	queryattempts
serverentry	—
servergroup	—
serverprincipal	—
syncinterval	syncinterval
tolerance	tolerance

Table 14 lists the server and clerk attributes that you cannot set.

Table 14 (Page 1 of 2). Unsettable DTS Object Attributes

Servers	Clerks
actcourierrole	—
autotdfchange	autotdfchange

Table 14 (Page 2 of 2). Unsettable DTS Object Attributes

Servers	**Clerks**
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 section in the *OS/390 DCE: Command Reference*.

DTS Timestamp Format

All responses to **dcecp** and **dtscp** commands contain a timestamp that conforms to the input and output format shown in Figure 51 on page 274.

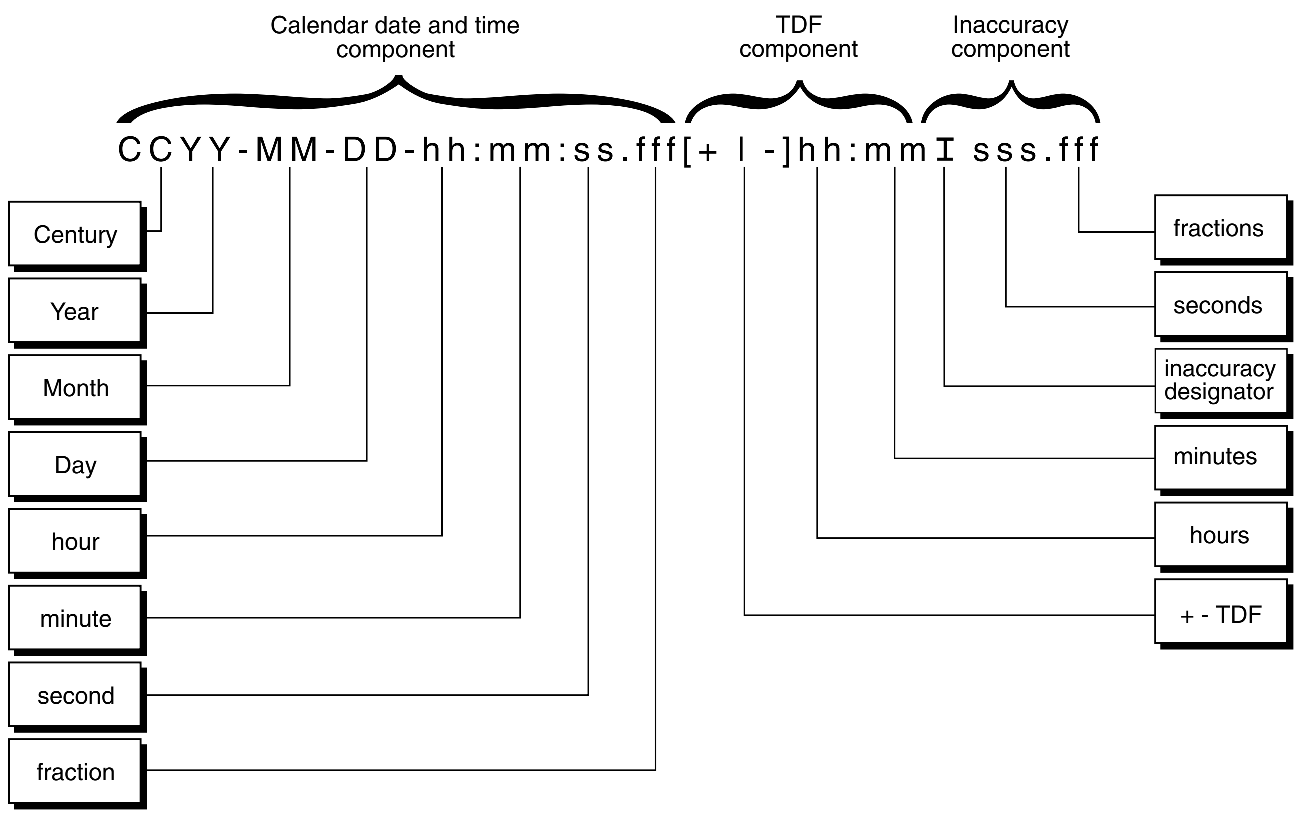


Figure 51. DTS Timestamp Format

The following example shows a typical DTS time display:

```
1996-03-16-14:29:47.52000-05:00I000.003
```

The timestamp uses the DTS format that is explained in Chapter 30, “Introduction to the DCE Distributed Time Service” on page 259. In this example, the year is 1996, 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.

Reconfiguring DTS on Nodes

DTS is initially configured during the overall DCE configuration procedure for a node. 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. If you choose to do this, you must perform the following steps:

1. Stop the clerk or server process (DTS daemon) that is currently executing on the node.
2. Run the DCECONF program to restart the DTS daemon on the node as a clerk or server.
3. Set any clerk or server attribute values as needed.

The following subsections provide detailed instructions for performing each of the reconfiguration steps listed above.

Stopping an Existing Clerk or Server

To stop the existing DTS clerk or DTS server on a node, use the **dcecp 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 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 functions of the DCECONF program that configure additional DTS clerks and servers. The DCECONF functions for configuring additional clerks and servers restart the DCE daemon (**dtsd**) as either a clerk or server.

Just as during initial DTS configuration, if you are creating a server, you must tell the DCECONF program the type of server that it is to create: global or local. Before you choose the server type, you should 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.

"Designating Global and Courier Servers" on page 283 provides more information about server courier roles and instructions for changing the courier role after you create a server.

Setting Clerk and Server Attribute Values

After you have created a new clerk or server on a node, you will want to set certain of the entity's 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 preexisting servers in the network segment. You 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" on page 285.

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 preexisting servers. For instance, 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 checking for faulty 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 preexisting clerks and server's in the network.

General instructions for modifying the attributes of DTS clerks and DTS servers are covered in “Modifying Clerk and Server Attributes.”

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.

To display the attribute values of a DTS clerk or DTS server, you use the **dts show** command. (The **dts show** command can also be used to view the values of DTS entity counters, however you cannot modify counter values.

For example, to display the attributes values for the DTS entity 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}
```

The example display 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, there will be more attributes displayed for a server (see Table 13 on page 272 and Table 14 on page 272).

If you wish to modify the attributes for a DTS clerk or server, you can use the **dcecp dts modify** command. Several examples of this command appear in the following subsections, which describe the settable attributes for clerks and servers. These subsections also offer suggestions for various attribute settings, depending on your network configuration.

The minservers Attribute

The **minservers** attribute specifies how many servers must supply time values to the system before DTS can synchronize the local 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 to set the desired value. The command accepts values from **1** to **10**. For example, to increase the required number of servers to **4**, enter the following command:

```
dcecp> dts modify -change minservers 4
```

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 both or either of these attributes, you can 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. 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 00000126-4769-21cf-8400-08005a191a6c}
{localservers
 {name /.../cellname/hosts/host1/self}
 {timelastpolled 1996-01-11-16:19:49.011+00:00I23.822}
 {lastobstime 1996-01-11-16:19:49.282+00:00I25.182}
 {lastobsskew +0-00:00:00.271I49.003}
 {inlastsync TRUE}
 {transport RPC}}
{localservers
 {name /.../dcecell14.endicott.ibm.com/hosts/DCECDS3/self}
 {timelastpolled 1996-01-11-16:19:49.011+00:00I23.822}
 {lastobstime 1996-01-11-16:19:49.439+00:00I25.333}
 {lastobsskew +0-00:00:00.428I49.155}
 {inlastsync TRUE}
 {transport RPC}}
```

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 event 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 Attribute 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. (See “Advertising Global Servers” on page 283 for further information.)

The presence of global servers in your network can influence the value that you choose for the **minservers** attribute. 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. 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 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 **dts show** displays the name, node ID, and node name for all of the global servers known by the local node.

Use of 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 3. 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 allowable 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 vary 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 the following:

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 that 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” 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 in the following cases:

- If a time provider is not used in the network
- If a system is part of a WAN-only network configuration
- If the applications that call DTS do not require the level of precision achieved by the default setting

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
```

The syncinterval Attribute

The **syncinterval** attribute prevents your system from synchronizing more often than the specified interval. This attribute prevents the **maxinaccuracy** attribute from causing continuous synchronizations. As mentioned in “The maxinaccuracy Attribute” on page 279 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 specified interval value. (The **syncinterval** attribute 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 that if the system reaches the **syncinterval** attribute setting, but has not yet reached the **maxinaccuracy** attribute setting, the system does not synchronize.

The default **syncinterval** attribute value is 2 minutes for servers and 10 minutes for clerks. If you are trying to minimize the skew between systems, you can lower the **syncinterval** attribute value. For example, 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 00-00:05:00.0000
```

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 only affects automatic synchronizations triggered by the **maxinaccuracy** attribute. (See the *OS/390 DCE: 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 the following:

- Defects in the clock hardware, including clock drift that is greater than the manufacturer's specifications.
- Malfunctioning time providers.
- Hardware clock ticks are lost by the operating system.
- 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 do not intersect with those of the servers used for synchronization. This process is shown in Figure 52, where value **t2** is faulty.

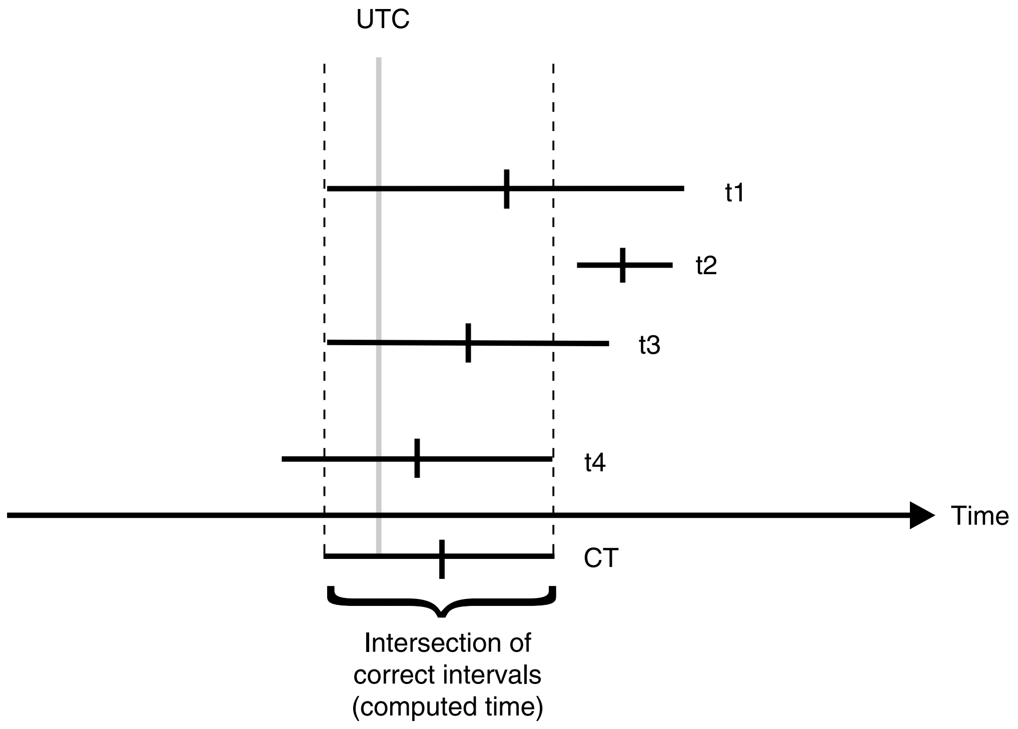


Figure 52. Local Fault

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 the 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, you will want to 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 an irrecoverable error. If you could set the **tolerance** attribute value 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 following example shows how to set the **tolerance** attribute value to 3 minutes:

```
dcecp> dts modify -change tolerance 00-00:03:00.000
```

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. After the time-out setting has elapsed the number of times that is determined by the **queryattempts** attribute, the system quits 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 following example shows how to set the **queryattempts** attribute value:

```
dcecp> dts modify -change queryattempts 4
```

The default setting for the **localtimeout** attribute is 5 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 because of the variations in WAN topologies and transmission quality. In the following example, the **globaltimeout** setting is changed to 20 seconds:

```
dcecp> dts modify -change globaltimeout 00-00:00:20.000
```

If you continually receive `Server Not Responding` event reports for a global server, increase the **globaltimeout** 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 Security Service. If you subsequently wish to change the name of a server, you can do this by changing two of the server's attributes: the **serverentry** attribute and **serverprincipal** attribute. The default settings for these **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 following is an example of a command that sets the **serverentry** attribute value:

```
dcecp> dts modify -change serverentry hosts/cyclops/dts_ref_node
```

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 Chapter 37, “Creating and Maintaining Accounts” on page 347 for further information on creating a new principal account and see Chapter 36, “Creating and Maintaining Principals, Groups, and Organizations” on page 331 for adding a principal name to an existing server group.

Note: The **serverprincipal** attribute is a read-only attribute on OS/390.

The following example command sets the **serverprincipal** attribute:

```
dcecp> dts modify -change serverprincipal hosts/ajax/dts_machine
```

Management Tasks Specific to Servers

Managing DTS servers involves some special tasks. These tasks include the following:

- Setting a server's epoch
- Assigning the courier role to a server
- Designating a server as a global server
- Setting the attributes for a connection to a time provider

The following subsections describe the server-specific tasks identified above.

Designating 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. To assign server roles, follow the instructions in the following subsections.

Advertising Global Servers: To assign a server to the global set of servers, you must advertise the server with the **dcecp 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. Because 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.

The following command 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.

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), as follows:

- **backup**
- **courier**
- **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 that 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.

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 command:

```
dcecp> dts modify -change courierrole backup
```

Designating Global Servers Outside a Cell

Advertising a global server adds the server's entry to the cell profile, where it is available to the other servers and clerks within that cell. To allow servers to access global servers in a different cell, you must manually add the combined remote cell/server name to the cell profile. The **dcecp dts configure** command cannot perform this function.

To manually add a global server name from a remote cell to the local cell profile, follow these steps:

1. Use the **dcecp rpcprofile show** command to display the cell profile for the remote cell as shown in the following example. You must know the name of the cell or obtain it from the cell administrator.

   ```
   dcecp> rpcprofile show /.../remote_cell/cell-profile
   ```

   After you enter this command, the remote cell's global server names and their matching UUIDs are displayed. Select and record the name of the global server you want to add to the local cell profile.

2. Use **rpcprofile add** command to add the global server name you want to add to the local cell profile, as in the following example:

   ```
   dcecp> rpcprofile add cell-profile -member /.../remote_cell/hosts/cliburn \
   -interface 1757914-82c9-11c9-8a59-08002b0dc035,1.0
   ```

3. After you have added the global server name from the remote cell to the local cell profile, you must add the principal name of the server to the **/.:/subsys/dce/dts-servers** security group using the

dcecp group add command. (You must have the appropriate permissions to edit the registry database). The following command adds the server:

```
dcecp> group add /.:/subsys/dce/dts-servers -member svr_1
```

See the **group** command in the *OS/390 DCE: Command Reference* for further information on adding groups to the registry.

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 server 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 preexisting servers have matching epoch numbers. Enter the **dcecp dts show** command to find out the epoch number of the server. For example:

```
dcecp> dts show /.:/hosts/orion/dts-server
```

Examine the attributes list that the command returns for the server's **epoch** attribute value. If the epoch of the 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 preexisting servers. The following example shows how to change a server's epoch number after you enable the server:

```
dcecp> clock set -abruptly -epoch 0
```

After you know that a server is starting up with the proper epoch number, *do not* change the epoch unless serious system or network problems damage all of the server clock values. In the unlikely event that the majority of the server clocks become faulty, use the **dts show** and **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

If a server is connected to a time provider, set its **checkinterval** attribute. 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, note that 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. The following example shows how to set the **checkinterval** attribute value:

```
dcecp> dts modify /.:/hosts -change checkinterval 00-00:00:30.0000]
```

Changing the System Time

There are three ways you can change system's time by using **dcecp** program commands. The following subsections describe 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.

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 clock set** command when you want 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 form 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.

Because it is a manually entered command that modifies 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 new time and then begin monitoring the reference. When you perceive that the reference has reached the desired time, you press <**Enter**> to start the command. Your perception of the reference mark and your pressing of <**Enter**> do not exactly coincide. Furthermore, after the command is initiated, DTS takes time to interpret and run the command.

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 1995-10-07-09:30:15.00I01.00
```

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 **clock set** command with the **-abruptly** option immediately (nonmonotonically) changes the system clock setting to the specified time, rather than gradually (monotonically) adjusting the time.

Note: Exercise 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 usually 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 with the **-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 again share the same epoch. 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.

Because it is a manually entered command that modifies 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 new time and then begin monitoring the reference. When you perceive that the reference has reached the desired time, you press <**Enter**> to start the command. Your perception of the reference mark and your pressing of <**Enter**> do not exactly coincide. Furthermore, after the command is initiated, DTS takes time to interpret and run the command.

The following example shows how to change both the time and epoch for a system:

```
dcecp> clock set 1995-10-07-09:30:15.0000I01.0000 -abruptly -epoch 1
```

Forcing System Synchronization

After you create and enable DTS on all the systems that are 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 for the amount of time that is 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, you 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
```

Controlling Access to DTS

You can assign privileges that control access to DTS objects by using DCE Authorization Service Access Control Lists (ACLs).

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** 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 Chapter 34, “Using Access Control Lists” on page 307 for a discussion of the DCE ACLs facility and descriptions of ACL types and their entries.

To display the ACL entries in the DTS server principal's ACL, you can use the **dcecp acl show** command. For example:

```
dcecp> acl show /.:/hosts/Detroit2/dts-entity
{unauthenticated r--}
{user hosts/Detroit2/self  wc}
{group subsys/dce/dts-admin rwc}
{any_other r--}
```

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 Chapter 34, “Using Access Control Lists” on page 307.

Chapter 32. OS/390 DCE Considerations in DTS

This section describes the implementation-specific aspects of DTS in OS/390 DCE.

DCE Software Clock

In OS/390 DCE, DTS maintains its own software clock (the DCE software clock), which is initially based on the OS/390 hardware clock. DTS adjusts the DCE software clock, not the OS/390 hardware clock. Thus, the time obtained from the DCE software clock may be different from that obtained from the OS/390 hardware clock. In this book, **system clock** refers to the DCE software clock. Distributed applications must always use the DCE software clock to maintain a consistent distributed time across the network.

The DCE software clock consists of two parts:

- The S/370™ TOD clock value
- An offset from the TOD clock.

DTS adjusts the DCE software clock by adjusting the offset from the TOD clock. The DCE software clock is stored in GMT. If your TOD clock is not relative to GMT, then you must compensate this through the offset such that the DCE software clock is relative to GMT.

Null Time Provider

In OS/390 DCE, a null time provider program is provided that can be used by OS/390 DCE host systems that have been configured as a DTS server.

What Is the Null Time Provider Program?

A DTS server uses a time provider program as an interface to time devices that can acquire UTC time values from external time sources through radio, satellite, or telephone. DCE provides Application Programming Interfaces that can be used to develop time provider programs.

The **null time provider** is one type of time provider program. However, it is different from other time provider programs in that it does not actually obtain time from external sources. Rather, it assumes that the host system's time is a reliable source of UTC time. With the Null Time Provider program, DTS does not make any adjustment to the host system time.

The Null Time Provider program is available in OS/390 DCE and it runs as a daemon, called the **Null Time Provider** daemon. This daemon can be run only if a DTS server (global or local) entity has been configured on the host system.

The Null Time Provider daemon is useful in typical OS/390 configurations, such as in **Sysplex** environments where the OS/390 host system is already accessing reliable ETR time.

Note: You can also choose to use your own time provider program. In this case, before starting DCEKERN, you will have to manually edit the Daemon Configuration File, **/opt/dcelocal/etc/euvpdcf**, and enter the load module name of the program in the load module field of the file. You can also start the time provider program as a separate address space.

DTS and the OS/390 Sysplex Environment

In a typical Sysplex environment consisting of multiple OS/390 systems running on more than one processor, the processors are connected to an IBM 9037 Sysplex Timer® hardware facility. It provides a single external time source, and synchronizes the time across all the processors from one central OS/390 system.

If DCE runs on a host within a Sysplex environment with a Sysplex timer, the Null Time Provider daemon can be used to disable DTS synchronization on the OS/390 host. The Null Time Provider daemon reads the DCE Software Clock time and gives it to the DTS entity. The DTS entity then ignores all other times from other DTS servers and does not adjust the DCE software clock. The net effects of these are:

- DTS does not adjust the DCE software clock.
- The adjustment of the DCE software clock will then be controlled by the Sysplex adjustment of the TOD clock (from which the DCE software clock is based upon).

Figure 53 is an example of a Sysplex Timer-DTS configuration.

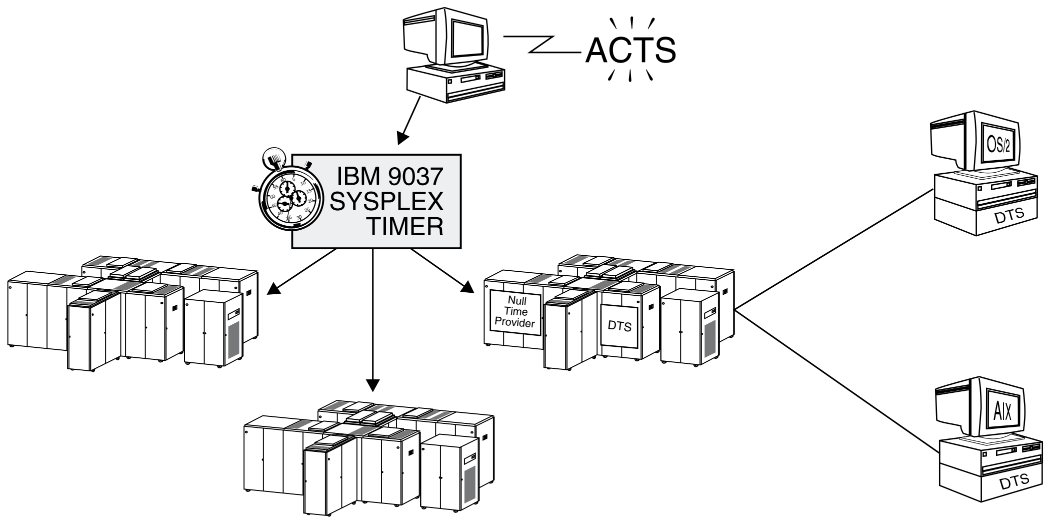


Figure 53. Sysplex Timer-DTS Configuration

The Null Time Provider is really a mechanism to make DTS use the already-trusted time provided by the Sysplex timer.

Because the OS/390 system that is part of the Sysplex is most likely also a DTS time server, it will provide its time to other DTS servers in the cell. As a result, the ETR time provided by the Sysplex timer will be disseminated to other requesting time servers or clerks in the cell.

Setting the Time Zone

The OS/390 DCE host system must be set to the proper time zone in order for DTS to display the correct local time. You can set the host system's time zone by setting the TZ environment variable to the correct POSIX time format value, such as EST5EDT. By default, TZ is set to GMT0.

Overriding the System Time Zone

Each user (including distributed applications running under their own identities) can override the system time zone (selected in the **localtime** file) by setting the **TZ** environment variable in the user's **envar** file or in the environment setting of the TZ variable. The **envar** file is located in the user's home directory. Any program that runs under the identity of that user utilizes the environment variables that are declared in the user's **envar** file.

By setting the **TZ** environment variable, the user designates his or her time zone, which allows programs to process time in the user's local time zone format.

The **TZ** environment variable can accept only a POSIX time format value.

Using POSIX Time Format in TZ

The POSIX time format is a character string that specifies the deviation of the geographical zone from the Greenwich Mean Time (GMT) and other information such as how to calculate the Daylight Savings Time. For example, to set the TZ environment variable to a time zone that is five hours ahead of GMT, the following TZ environment variable declaration can be made:

```
TZ=EST5EDT
```

where the fourth character of the value, **5**, indicates that it is deviating five hours from GMT. The characters **EST** and **EDT** are the designations for standard and summer time zones, respectively. This example sets time conversion information for Eastern Standard and Eastern Daylight Savings Time in the United States.

For more information on the POSIX format for **TZ** values, refer to *OS/390 DCE:Application Development Reference*.

Resetting the DCE Software Clock

The **MODIFY DCEKERN CLOCK** operator command can be used to reset the DCE software clock, as well as display information about it. Because this command is run outside of the DTS control program, it is especially useful if you have to reset the clock when the DCE daemons are not running. For example, during the configuration of the host system (using the **DCECONF** program), configuration will not succeed if the difference between the time of the host system and the security server host is more than five minutes. In this case, the MODIFY DCEKERN CLOCK command sets the DCE software clock to an acceptable value.

The MODIFY DCEKERN CLOCK operator command has the following command options for resetting the DCE software clock:

reset	Resets the DCE software clock to exactly the same value as that of the OS/390 hardware clock.
set	Sets the clock to a specified time. This requires a parameter in UTC absolute time format.
setrel	Adds the specified relative time to the DCE software clock.
setdst *N*	Sets the daylight savings time field of the clock, where *N* is an integer to which the daylight savings time flag is set.

The **display** command option displays information on the time zone, clock offset, and adjustment of the DCE software clock.

For more information on the MODIFY DCEKERN CLOCK command, see the *OS/390 DCE: Command Reference*.

Part 8. DCE Security Service

Chapter 33. Overview of the DCE Security Service . . . 297
DCE Authentication Servers and Clients . . . 297
The Registry Database . . . 298
  Cells . . . 299
The Logical Identities of Principals, Groups, and Organizations . . . 299
  Full Pathnames of DCE Objects . . . 299
  Full Pathnames for Registry Objects . . . 300
Physical Security of the Database . . . 300
How the Registry Database is Stored . . . 300
Replicated Databases . . . 301
How Updates Are Handled . . . 301
  Master and Slave Replicas . . . 302
  Handling Database Updates . . . 303
  Propagating Database Changes . . . 304
  Master/Slave Authentication . . . 304
Names for Security Objects . . . 304
  Using Names with dcecp Security Service Commands . . . 305
  Using Names with the dcecp acl Command . . . 305

Chapter 34. Using Access Control Lists . 307
Authorization Overview . . . 307
  ACL Managers . . . 308
  ACL Interpretation . . . 309
  Privilege Attributes Inherited by Processes 309
ACL Entries and Masks . . . 309
  ACL Syntax . . . 310
  ACL Entry Types for Principals and Groups . . . 310
  Group Permissions and Project Lists . . . 314
  Using Principal and Group ACL Entries . . 314
  ACL Entry Types for Masks . . . 315
  ACL Entry Types for Dissimilar DCE Releases . . . 315
  The Checking Sequence for ACL Entries . 316
  Denying Access . . . 319
ACL Management Tasks . . . 319
Copying ACLs . . . 320
Generating ACLs from Files . . . 320
Container ACLs . . . 321
  Objects and Containers . . . 321
  Initial ACLs for Objects and Containers . . 321
  Effect of Masks when Editing ACLs 324

Chapter 35. Control Programs for Managing the Security Service . . . 327
Using the DCE Control Program . . . 327
  Security Service Objects . . . 327
  Security Service Command Operations . . 328
Using the Registry Editor . . . 329
  Starting, Stopping, and Getting Help . . . 329
Using the sec_admin Program . . . 330
  Starting, Stopping, and Getting Help . . . 330
  The sec_admin Commands for Reconfiguring Replica Sets . . . 330

Chapter 36. Creating and Maintaining Principals, Groups, and Organizations . 331
Principal, Group, and Organization Names . 331
  Primary Names . . . 331
  Full Names . . . 331
  Aliases . . . 332
  Name Formats . . . 332
Reserved Principals and Accounts . . . 332
Object Creation Quotas . . . 333
Universal Unique Identifiers and UNIX IDs . 333
Adding and Maintaining Principals . . . 334
  Adding Principals . . . 334
  Changing Principals . . . 335
  Deleting Principals and Aliases . . . 336
Extended Security Attributes for Principals . 336
  OS/390 DCE Release 1 Authentication . . 336
  Managing Logins that are Not Valid 339
  Managing Password Strength and Password Generation . . . 339
Adding and Maintaining Groups and Organizations . . . 341
  Project Lists . . . 341
  Adding Groups and Organizations . . . 342
  Changing Groups and Organizations . . . 343
  Deleting Groups and Organizations 343
Maintaining Membership Lists . . . 344
  Effects of Account Creation on Membership Lists . . . 344
  Example: Adding and Deleting Group Members . . . 344
Creating and Maintaining Aliases for Principals or Groups . . . 345
  Creating Aliases . . . 345
  Changing Primary Names to Aliases and Vice Versa . . . 345

Chapter 37. Creating and Maintaining Accounts . . . 347
User Accounts . . . 347
Server Accounts . . . 347
  Passwords for Server Accounts . . . 348

Accounts for Servers 348
Steps for Creating Server Accounts 348
Machine Accounts 349
How Identities Represented by Accounts Are Authenticated 349
Privilege Attributes 349
Ticket-Granting Tickets and Tickets to Services 350
Displaying Privilege Attributes and Tickets .350
Destroying a Principal's Tickets 351
Adding Accounts 351
Setting Ticket Lifetimes 353
Adding Accounts Example 354
Modifying Accounts 355
Deleting Accounts 355
Reactivating Accounts that have Expired . 356
Creating, Maintaining, and Deleting Keytab Files 357
The Keytab File 357
Creating and Maintaining Keys and Keytab Files 358
Removing Keytab Files 360
Changing Server and Machine Passwords in the Keytab File 361
Handling Compromised Server or Machine Passwords in the Keytab File 361

Chapter 38. Creating and Using Extended Registry Attributes 363
The xattrschema Object 363
Creating and Maintaining Attribute Types . . 363
Creating Attribute Types 363
Modifying Attribute Types 365
Renaming Attribute Types 365
Deleting Attribute Types 365
Defining the ACL Managers for Attributes .366
Defining the Attribute Type Encoding . . . 367
Defining Attribute Trigger Servers 368
The trigtype Option 368
The -trigbind Option 369
Creating and Maintaining Attribute Instances .371
Attaching Attribute Instances to Objects . 371
Modifying Attribute Instances 371
Deleting Attribute Instances 372
Using Attribute Sets 373

Chapter 39. Administering a Multicell Environment 375
Trust Relationships 375
Direct Trust Relationships 375
Establishing Trust Relationships 376
Creating Trust Relationships 376
Command Options for the registry connect Command 377
The Accounts Created by the registry connect Command 378
Changing Cross-Cell Authentication Accounts 379

Chapter 40. Viewing Registry Information 381
Displaying Account Information 381
Displaying Group and Organization Information 382
Displaying Principal Information 384
Displaying xattrschema Information 385
Displaying ACL Information 386
Displaying keytab Information 387

Chapter 41. RACF Interoperability and Single Sign-on 389
Overview of RACF Interoperability 389
The RACF Interoperability Utilities 390
Tailoring the Utilities for Your Environment . 393
Tailoring Variables Common to mvsimpt and mvsexpt 393
Tailoring Variables for mvsimpt 393
Tailoring Variables for mvsexpt 394
Guidelines for Using mvsimpt and mvsexpt . 395
Multi-Cell Considerations 398
Cross Linking Small Numbers of Users . . 400
Recreating the Processed Entries File . . 400
Introduction to Administration Scenarios . . . 400
Cross Linking Existing RACF Users who are New DCE Users 400
Cross Linking Existing DCE Users who are New RACF Users 405
Cross Linking Existing DCE Users who are Existing RACF Users 407
Single Sign-on for OS/390 and DCE 409
Preparing for DCE Single Sign-on 409
Automatic DCE Single Sign-on Invocation .410
User Control of Automatic DCE Single Sign-on 410

Chapter 42. Maintaining Policies and Properties 411
Policies 411
Standard Policy 411
Authentication Policy 413
Handling Conflicting Policies 414
The Effects of Changes on Existing Policies 414
Displaying and Setting Standard and Authentication Policies 415
Properties 415
Default Ticket Lifetime Property 415
Hidden Password Property 415
Minimum Group ID Property 416
Minimum Organization ID Property 416

Minimum UNIX ID Property 416
Maximum UNIX ID Property 416
Minimum Ticket Lifetime Property 416
Displaying and Setting Properties 417

Chapter 43. Performing Routine Maintenance 419
Adding Accounts 419
Changing the Registry's Master Key 419
Validating the Authenticity of the DCE Security Service 420
Backing Up and Restoring the Registry Database 420
Procedures for Backing Up the Registry Database 420
Procedure for Restoring the Registry Database 421
Setting the _s(sec) Variable 422

Chapter 44. Handling Network Reconfigurations 423
Changing the Master Replica Site 423
Removing a Server Machine from the Network 424
Handling Network Address Changes 424
Updating the pe_site File 424
Handling Simultaneous Address Changes .425

Chapter 45. Adopting Registry Orphans . 427
What are Orphans? 427
Solving the Problem of Orphans 427

Chapter 46. Accessing Registry Objects . 429
The Registry Database 429
Registry Permissions 430
Management, Authentication, and User Information 430
Permissions to Create Principals, Groups, or Organizations 432
Permissions to Delete Principals, Group, or Organizations 433
Permissions to Add Accounts 433
Permissions to Delete Accounts 436
Permissions to Add Members to Groups . 437
Permissions to Add Members to Organizations 437
Permissions to Delete Members from Groups or Organizations 437
Permissions to Change a Principal's, Group's, or Organization's Full Name . . 438
Permissions to Change Management Information for Principals, Groups, or Organizations 438
Permissions to Change Management, Authentication, and User Information (Except Passwords) for Accounts 439
Permissions to Change Passwords for Accounts 439
Permissions to Change Authentication and Management Information for Registry Policies and Properties 439
Permissions to Run Commands that Act on Replicas 440
Permissions to Create Extended Registry Attribute Types 440
Permissions to Delete Extended Attribute Types 440
Permissions to Modify Extended Registry Attribute Types 441
Permission to Change ACLs on Registry Objects 441
Permissions Required by Slave Replicas . 442
Registry ACL Manager 442
Initial Registry ACLs 443

Chapter 47. DCE Audit Service 445
Features of the DCE Audit Service 445
Components of DCE Audit Service 445
DCE Audit Service Concepts 445
Audit Clients 446
Code Point 446
Audit Event 446
Event Number 446
Event Class 446
Event Class Files 447
Event Class Names 447
Event Class Numbers 447
Event Class Number Formats 448
Filters 448
Audit Trail File 451
Administration and Programming in DCE Audit 451
Programmer Tasks 452
Administrator Tasks 453

Chapter 48. DCE Audit Service Administrative Tasks 457
Setting DCE Audit Environment Variables . . 457
Starting the Audit Daemon 457
Controlling Access to the Audit Daemon . . . 458
DCE Permissions Supported by the DCE Audit Service 458
Initial ACL of the Audit Daemon 458
Giving Permissions to Audit Clients and Administrators 458
Defining Event Classes 459
Steps in Defining an Event Class 459

Example Event Class File 459
Creating and Maintaining Filters 460
How To Create Filters 460
How To Modify Filters 461
How To Delete Filters 461
Default Filters 461
Enabling Audit Filters 462
Enabling and Disabling the Audit Logging Service 463
Modifying and Querying Audit Daemon Attributes 463
Controlling and Displaying Audit Trails 463
Displaying Audit Trail Files 463
Controlling the Audit Trail Size 464
Changing the Audit Trail File Storage Option 464

Chapter 49. Hardware Cryptography in DCE 465

Chapter 33. Overview of the DCE Security Service

This chapter briefly introduces the DCE Security component. The Security component consists of the following services:

- The **Registry Service**

  Maintains the registry database, which is a database of principals, groups, organizations, accounts, and administrative policies.

- The **Authentication Service**

  Handles user authentication or the process of verifying that principals are who they say they are. 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 the service.

- The **Privilege Service**

  Supplies the user's **privilege attributes** that ensure a principal has the privileges to perform requested operations.

  In addition DCE Security provides:

  – **Access Control List (ACL) Facility**

    Establishes and grants access rights to an object based on the object's access permissions.

  – **Extended Registry Attribute (ERA) Facility**

    Provides tools to extend the registry database schema to define additional attributes. It also provides tools to attach those attributes to registry objects.

  The DCE host daemon (**dced**) acts as the Security client.

  The Registry, Authentication, and Privilege Services exist as a single daemon, the security server (**secd**).

  Note: Although the OS/390 Security Server is not part of OS/390 DCE Base Services, it *is* available as an optional feature of OS/390. The management interfaces (**dcecp**, Registry Editor, and ACL Editor) are part of the OS/390 DCE Base Services.

  This part of the book primarily describes how to use the tools and commands used to create, access, change, and delete information in the registry database, and how to perform administrative maintenance functions on the registry.

DCE Authentication Servers and Clients

The Authentication Service consists of the registry database, security server, and security clients. The server accesses the database to perform queries and updates and to validate user logins. A security client communicates with a security server to request information and operations. 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. See Figure 54 on page 298 for a simplified representation of the relationship between registry clients, servers, and a registry database.

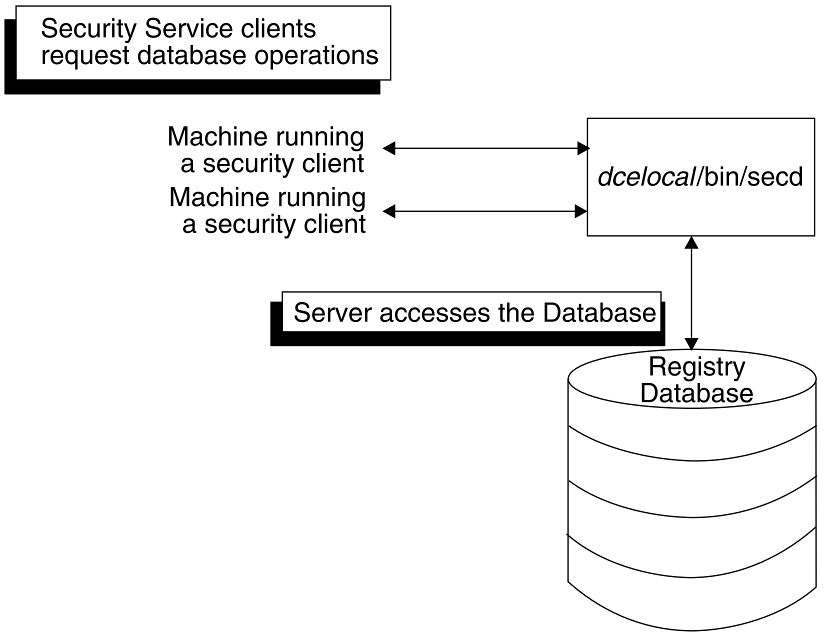

Security Service clients request database operations
Machine running a security client
Machine running a security client
dcelocal/bin/secd
Server accesses the Database
Registry Database

Figure 54. Machines, Servers, and the Database

The Registry Database

The registry database contains the following information:

Principals	Users of the system. Principals can be interactive principals (human users) or non-interactive (servers, machines, and cells). Principals can be associated with access permissions.
Groups	Collections of principals identified by a group name. Groups can be associated with access permissions.
Organizations	Collections of principals identified by an organization name. Organizations cannot be associated with access permissions.
Accounts	Contain the passwords and accounting information and allow principals authenticated access to objects within the cell. (Authenticated access can also occur between principals in different cells. See "Cells" on page 299 later in this chapter.)
Policies and Properties	Policies and properties regulate such things as password length and format and certain authentication requirements.
The replist Object	This object manages replicas of the registry database.
The xattrschema Object	This object is the extended registry schema created with the ERA facility.

(See Chapter 46, "Accessing Registry Objects" on page 429 for a detailed description of the structure of the registry database and the types of information it contains.)

The registry database also contains information on policies and properties for the registry as a whole and for organizations. Policies and properties regulate such things as password length and format and certain authentication requirements. Chapter 46, "Accessing Registry Objects" on page 429 describes the structure of the registry database in detail and the types of information it contains.

Cells

The collection of principals, groups, organizations, and accounts controlled by a registry database is an entity known as a **cell**. Authenticated communication is possible between cells only if those cells have special accounts in the registry database at the cell they want to communicate with. These special accounts set up cross-cell authentication, which allows principals from one cell authenticated access to another cell. For more information on cross-cell authentication, see Chapter 39, “Administering a Multicell Environment” on page 375.

The Logical Identities of Principals, Groups, and Organizations

In the Security Service, principal names, which are the names users enter when they login to DCE, represent a principal's logical identity. It is this logical identity that is authenticated by the Authentication Service, and it is the principal name that you enter when you log in. Principal names take the form:

[/.../cellname/]principal_name.

You are not required to enter ***/.../cellname/***. It defaults to the local cell.

Although there is no relationship required between a principal's logical identity and any object in the namespace, frequently such a relationship may be desired and can be enforced. In the Security Service there is a corresponding object in the registry database for each principal name. The database object, which is created automatically when a principal is created, contains attributes of the principal, such as account information.

Similarly, group names and organization names represent the logical identities of groups and organizations and take the forms:

[/.../cellname/]group_name

and

[/.../cellname/]organization_name

respectively.

Full Pathnames of DCE Objects

All DCE objects, including the object in the registry database that corresponds to each principal's logical identity, are identified by a full pathname. The full pathname, which is registered with the Cell Directory Service, takes the form /.../*cellname*/*object_name* where:

cellname	Is the name of the cell in which the object resides.
object_name	Can be the name of a simple object or a container object, such as a directory. If an 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.

Full Pathnames for Registry Objects

For registry objects, the object name is composed of:

- A mount point name that is the name under which the Security component is registered in the Directory Service.
- A directory path in the registry database that must be traversed to access the registry object.
- The name of the object itself.

The registry database contains three main directories: one each for principals, groups, and organizations. Each of the main directories can contain subdirectories. The registry object name is the name of the object (the name of a principal, a group, or an organization). If the object is a principal, its object name is the same as *principal_name* in the principal's logical identity. Similarly, if the object is a group or an organization, its object name is the same as the group or organization name in the group's or organization's logical identity.

For example, the full pathname for the principal **bach**, who resides in the cell **leipzig.com** which uses the **sec** (Security) mount point is

/.../leipzig.com/sec/principal/bach

The principal **bach** is an entry in the **principal** directory of the registry database in the cell **leipzig.com**.

As another example, assume the group **east-west** resides in **sales**, a subdirectory of the **group** directory in the registry database in the **dresden.com** cell. The full pathname for **east-west** is:

```
/.../dresden.com/sec/group/sales/east-west
```

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 communication. The security server and the database 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 authorized administrators, you need to 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 and by whom the server machine is accessed, and any other methods appropriate to your needs.

See the Security part of the *OS/390 DCE: Application Development Guide: Core Components* for a detailed discussion of authentication.

How the Registry Database is Stored

Each security server maintains a working copy of the registry database in virtual storage and a permanent copy on disk. All reads and updates operate on the copy in virtual memory. The servers use the copy on disk to initialize the copy in virtual storage when they start up. An atomic update log guarantees the database state in the event of server error.

Figure 55 on page 301 shows the server and the disk and virtual storage copies of the database.

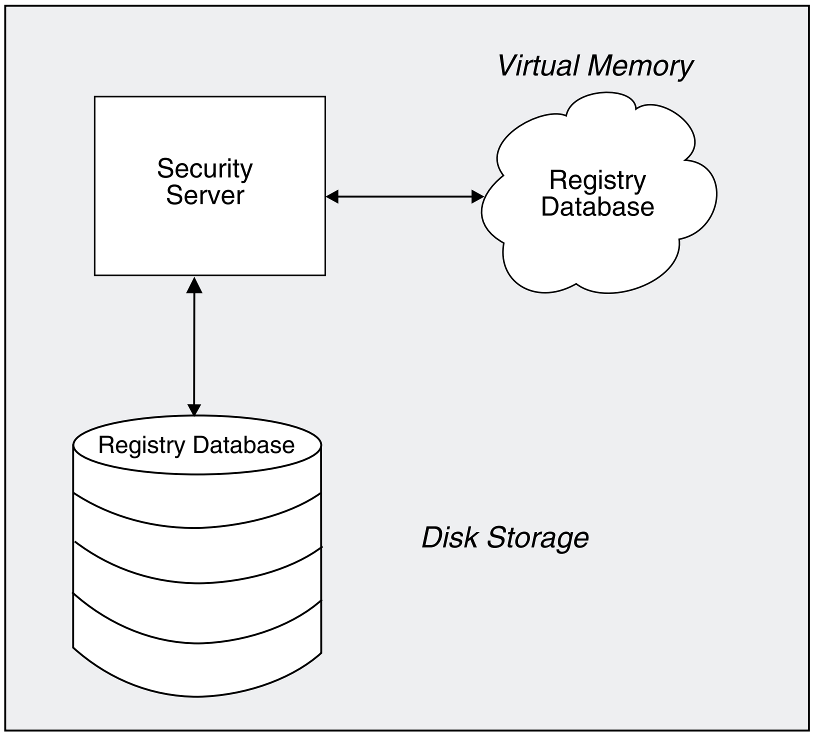


Figure 55. Disk Memory and Virtual Memory Copies of the Registry Database

Each security server periodically saves its entire database from virtual storage to disk. The database is stored in **/opt/dcelocal/var/security/rgy_data**.

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. The combination of a security server and its data (the registry database) is referred to as a **replica**. Typically, you 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.

How Updates Are Handled

Updates are made to only one database and the changes are propagated to all others. 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 slave replicas. Figure 56 and Figure 57 on page 303 illustrate the master and slave update processes. The processes are described in the sections that follow the figures.

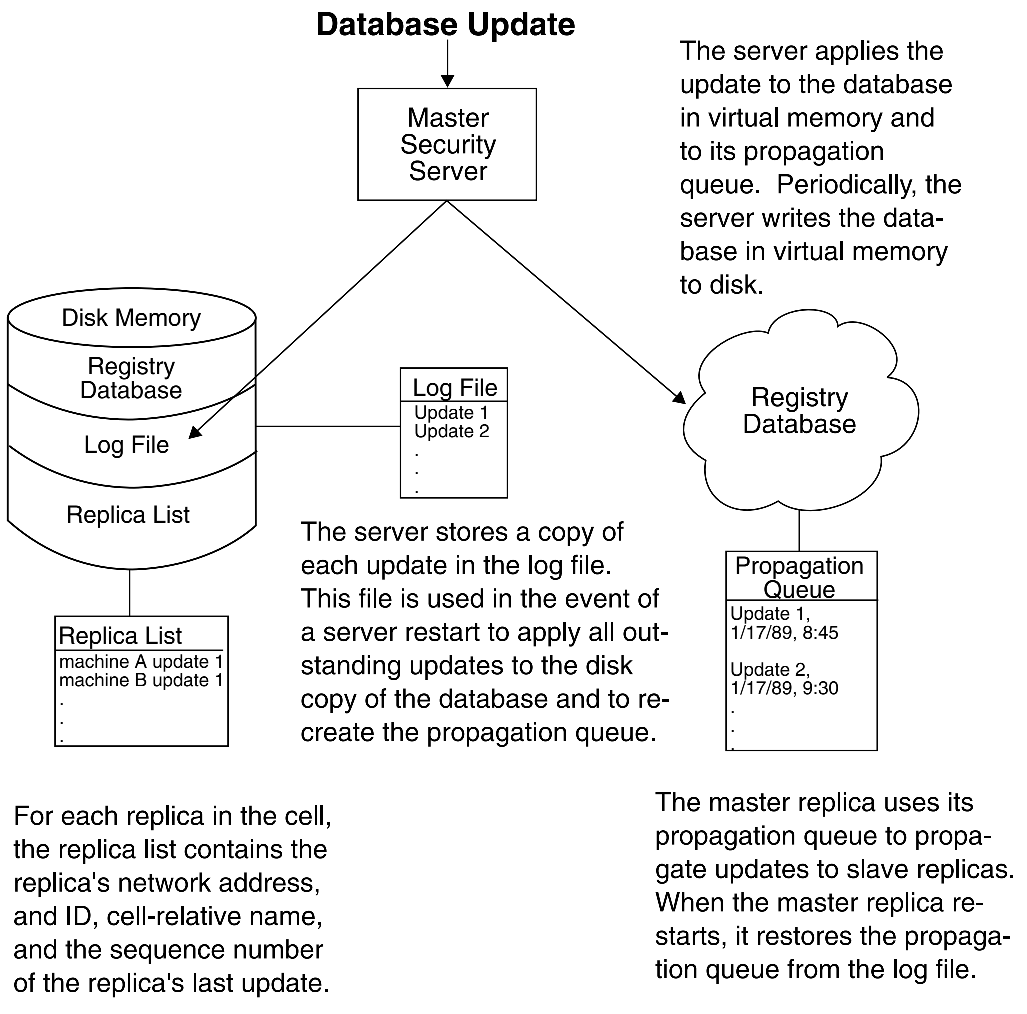

Database Update
Master Security Server
The server applies the update to the database in virtual memory and to its propagation queue. Periodically, the server writes the database in virtual memory to disk.
Disk Memory
Registry Database
Log File
Replica List
Log File
Update 1
Update 2
Registry Database
Propagation Queue
Update 1, 1/17/89, 8:45
Update 2, 1/17/89, 9:30
Replica List
machine A update 1
machine B update 1
The server stores a copy of each update in the log file. This file is used in the event of a server restart to apply all outstanding updates to the disk copy of the database and to recreate the propagation queue.
For each replica in the cell, the replica list contains the replica's network address, and ID, cell-relative name, and the sequence number of the replica's last update.
The master replica uses its propagation queue to propagate updates to slave replicas. When the master replica restarts, it restores the propagation queue from the log file.

Figure 56. The Master Replica Update Process

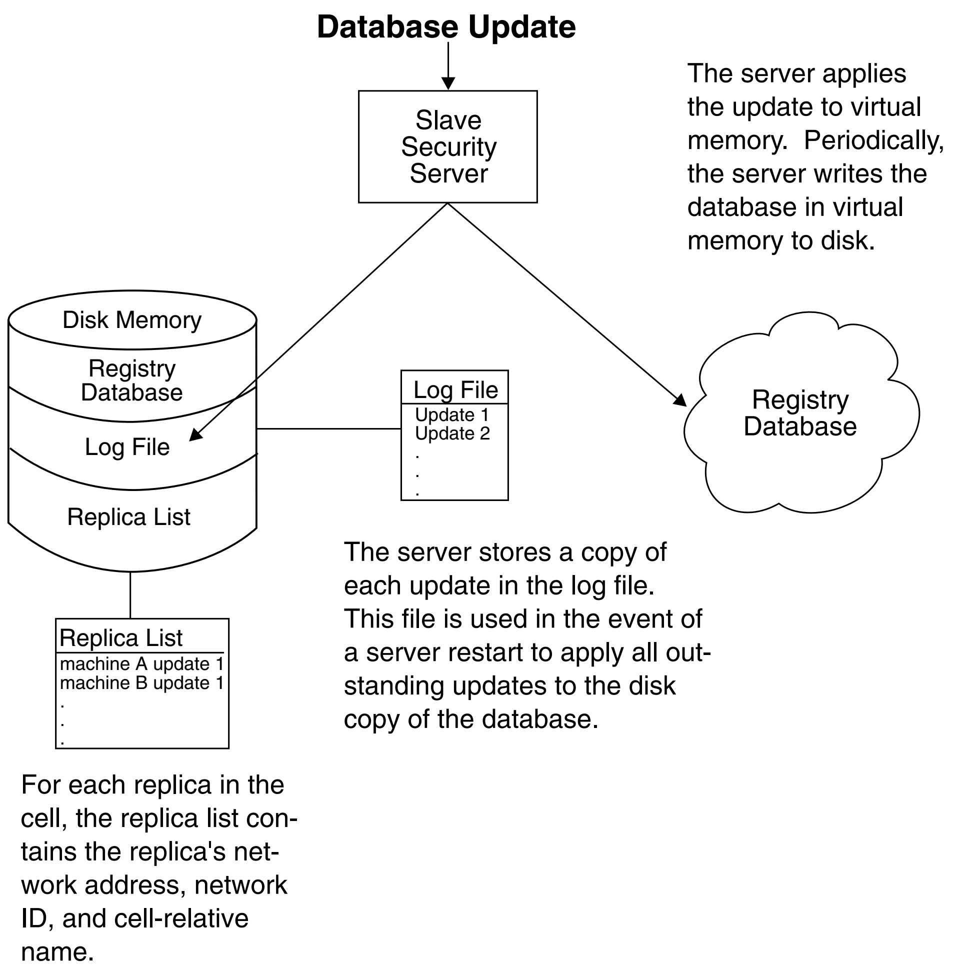

Database Update
Slave Security Server
The server applies the update to virtual memory. Periodically, the server writes the database in virtual memory to disk.
Disk Memory
Registry Database
Log File
Replica List
Log File
Update 1
Update 2
Registry Database
The server stores a copy of each update in the log file. This file is used in the event of a server restart to apply all outstanding updates to the disk copy of the database.
Replica List
machine A update 1
machine B update 1
For each replica in the cell, the replica list contains the replica's network address, network ID, and cell-relative name.

Figure 57. Slave Replica Update Process

Handling Database Updates

When a master or slave replica receives updates, it applies the update to its database in virtual storage, and saves a copy of the update in a **log file** stored on 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.

Periodically, the replica writes the database in virtual storage 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**, used to hold changes to be propagated to the slave replicas, as described in “Propagating Database Changes” on page 304. When the master replica receives an update, it adds it to the propagation queue in addition to its virtual storage 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 storage 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 using the process described in “Handling Database Updates” on page 303. Then, the master 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 associated with a replica in its replica list, and the sequence numbers of the updates 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. When the update succeeds, the master updates the sequence number 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/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 created when the database is created. Slaves carry the identity of the host machine on which they exist. Note that this identity must have the rights to the **/.:/sec/replist** object described in Chapter 46, “Accessing Registry Objects” on page 429.

Names for Security Objects

Because the Security namespace is rooted in the Cell Directory Service (CDS) namespace, Security objects have CDS pathnames, 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 generally registry objects reside in the principal, group, or organization directory in the registry database. See Chapter 46, “Accessing Registry Objects” on page 429 for a more complete description of the registry database structure.

For example, the full pathname for the principal **bach**, which resides in the cell **leipzig.com**, uses the **sec** (Security) mount point, and is in the **principal** directory, is as follows:

/.../leipzig.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 pathname for **east-west** is:

```
/.../dresden.com/sec/group/sales/east-west
```

Using Names with dcecp Security Service Commands

For all the **dcecp** commands that 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 **dcecp** 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 CDS pathname instead of just *object_name*.

For example to use the **dcecp acl** command to change the ACL associated with principal **bach**'s registry account, you must enter the following fully qualified name:

```
/.../leipzig.com/sec/principal/bach
```

or

```
/.:/sec/principal/bach
```

Note also that to use **dcecp acl** to manipulate the ACL that is 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:

```
/.../leipzig.com/sec/principal
```

Chapter 34. Using Access Control Lists

You can control access to DCE objects using an authorization mechanism called an Access Control List (ACL). ACLs are associated with files, directories, Cell Directory Service (CDS) entries, and registry objects. They can be used 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 use the object
- What kinds of access those principals or groups have to the object
- What kind of access is allowed to unauthenticated users

This chapter:

- Provides an overview of ACLs.
- Describes the form and purpose of ACL entries and masks, including the sequence in which entries are checked to derive permissions.
- Describes how to use the DCE control program (**dcecp**) to display, create, modify, and delete ACL entries; to use masks; to copy ACLs; and to edit different types of ACLs.

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

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

Authorization Overview

An ACL contains a list of entries that specify the principals who can access an object and the operations those principals can perform. The principals can be named specifically 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

- 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 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 58 shows ACL Managers in servers.

The client side lets you 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.

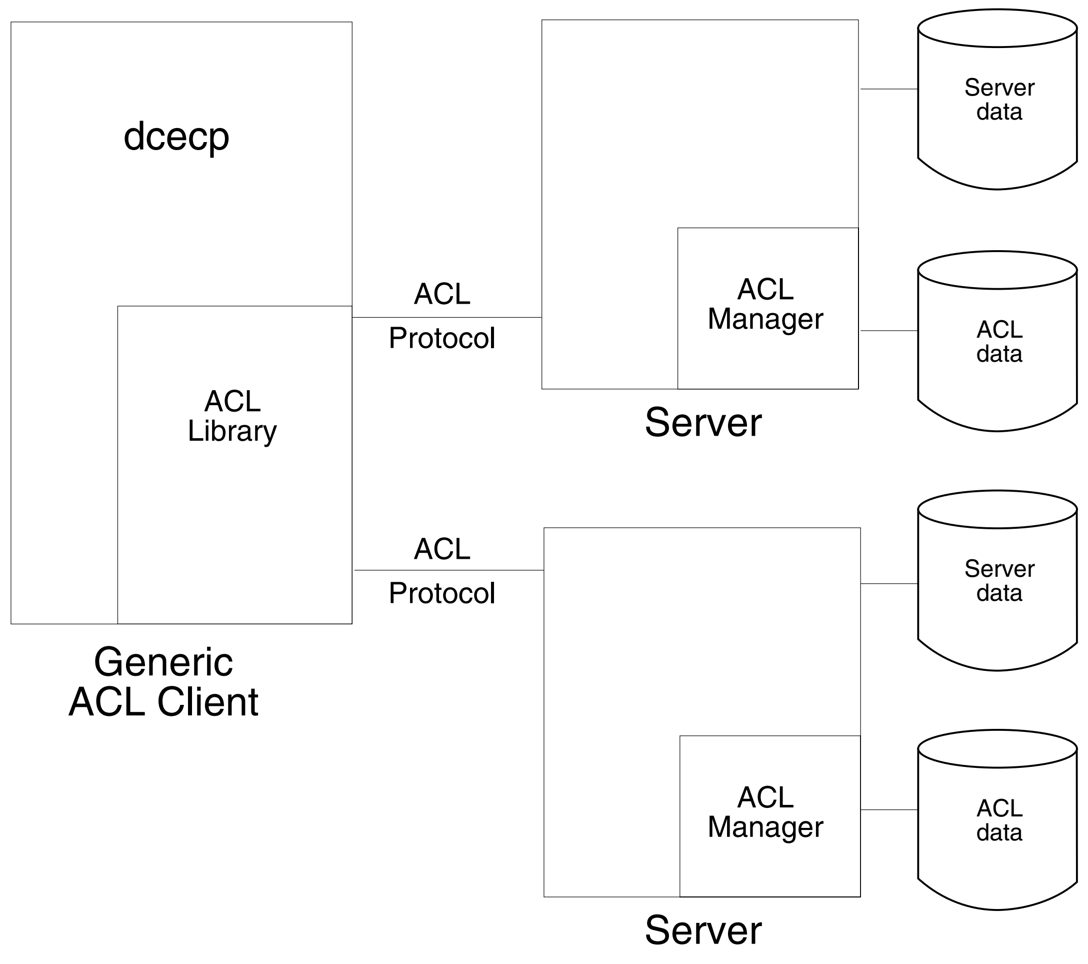


Figure 58. ACL Managers in Servers

In addition to the standard DCE components, ACLs can control access to any object for which an ACL Manager has been defined. 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 chapter are available to ACL Managers; however, each Manager may use 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 guide.

ACL Interpretation

Part of the information associated with an account is the principal name and the group (or groups) associated with the principal name in the account. (The groups are called a **project list** in this context.) 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 servers 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 of 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 is given access; if not, access is denied.

Privilege Attributes Inherited by Processes

Processes created or spawned by a principal inherit the principal's privilege attributes (or credentials). For example, if you log in, are authenticated, and start an application, the application you start inherits your authenticated privilege attributes 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 permissions and pass them 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 performs the equivalent of a user login, receives its privilege attributes, and runs under its own identity, not yours.

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.

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***}**

The three sample ACL entries in Figure 59 are in the format that Tcl accepts for input.

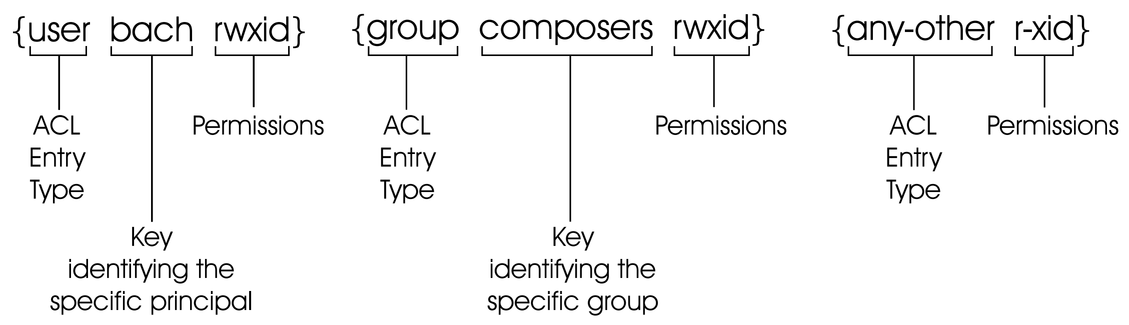


Figure 59. Sample ACL Entries

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 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 whose privilege attributes 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 chapter). 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 chapter).

The following table lists 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 table uses the following syntax variables:

- *principal_name* is the name of a principal in the registry database.
- *group_name* is the name of the group defined in the registry database.
- *cell_name* is the global pathname of a cell in the format **/.../.name**, where **/.../** is the DCE global prefix.
- *permissions* are the permissions made available by the object's ACL Manager.

Table 15 (Page 1 of 3). ACL Entry Types for Principals and Groups

ACL Entry Type	Purpose	ACL Entry Format
user_obj	Establishes permissions for the object's real or effective user. An example is the owner of a file.	{**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.	{**group_obj** *permissions*}
other_obj	Establishes permissions for all others 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.	{**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.	{**user** *principal_name permissions*}

Table 15 (Page 2 of 3). ACL Entry Types for Principals and Groups

ACL Entry Type	Purpose	ACL Entry Format
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.	**{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.	**{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.	**{foreign_group** *cell_name/group_name permissions***}**
foreign_other	Establishes permissions for other principals in a specific foreign cell 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.	**{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.	**{any_other** *permissions***}**
user_obj_delegate	Establishes permissions for an intermediary acting for the object's real or effective user.	**{user_obj_delegate** *permissions***}**
group_obj_delegate	Establishes permissions for an intermediary acting for members of the object's real or effective group.	**{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.	**{other_obj_delegate** *permissions***}**

Table 15 (Page 3 of 3). ACL Entry Types for Principals and Groups

ACL Entry Type	Purpose	ACL Entry Format
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.	**{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.	**{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.	**{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.	**{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_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.	**{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.	**{any_other_delegate** *permissions***}**

Group Permissions and Project Lists

Principals accrue group permissions from their project list, a list of all the groups in 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 Chapter 36, “Creating and Maintaining Principals, Groups, and Organizations” on page 331 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 Chapter 36, “Creating and Maintaining Principals, Groups, and Organizations” on page 331 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, you may think of the **user_obj**, **group_obj**, and **other_obj** types as similar to the **owner**, **group**, and **other** types for which UNIX permission bits are set. 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. When the Manager finds a match between a principal and an entry, it stops examining the ACL and applies the entry that is found (or in the case of groups, entries that are found). All other ACL entry types, except for mask types, are examined by the ACL Manager to see if a match exists before the ACL Manager examines the **any_other** entry type. ACL checking will be described later in this section. See “The Checking Sequence for ACL Entries” on page 316 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 in 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 (entry type **mask_obj**), if it exists, applies to all entry types except **user_obj** and **other_obj**. The unauthenticated mask (entry type **unauthenticated**) 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**, 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 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.

Here is an example of mask usage. 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 use 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 320 tells how to copy extended entries.

Note that extended entries cannot be changed; 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	The ACL entry type.
uuid	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	A Network Data Representation (NDR) format label (in hexadecimal format and separated by dots) that identifies the encoding of data.
number_of_bytes	A decimal number that specifies the total number of bytes in *data*. It is followed by a dot.

data	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	The permissions to be granted by the entry.

The Checking Sequence for ACL Entries

When an ACL Manager reads through the list of ACL entries, it 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 the 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 60 on page 318 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” on page 314 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 the ACL Manager 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 60 on page 318 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.

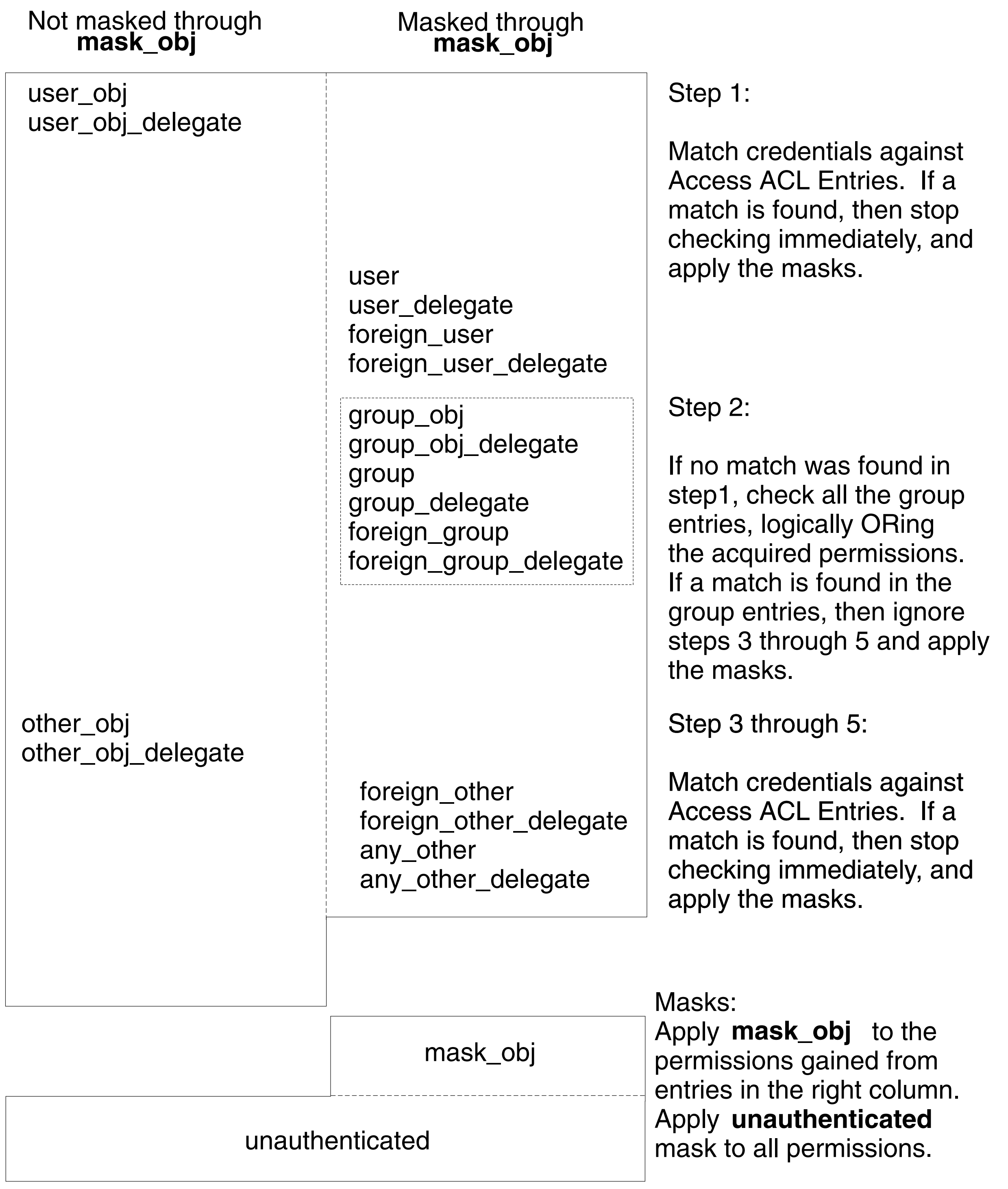

Not masked through **mask_obj**
Masked through **mask_obj**
user_obj
user_obj_delegate
Step 1:
Match credentials against Access ACL Entries. If a match is found, then stop checking immediately, and apply the masks.
user
user_delegate
foreign_user
foreign_user_delegate
group_obj
group_obj_delegate
group
group_delegate
foreign_group
foreign_group_delegate
Step 2:
If no match was found in step1, check all the group entries, logically ORing the acquired permissions. If a match is found in the group entries, then ignore steps 3 through 5 and apply the masks.
other_obj
other_obj_delegate
Step 3 through 5:
foreign_other
foreign_other_delegate
any_other
any_other_delegate
Match credentials against Access ACL Entries. If a match is found, then stop checking immediately, and apply the masks.
mask_obj
unauthenticated
Masks:
Apply **mask_obj** to the permissions gained from entries in the right column.
Apply **unauthenticated** mask to all permissions.

Figure 60. 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** permissions, 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**. After a matching user entry is found, checking stops and the corresponding 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 is:

```
{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.

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 programs **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.)
- Display the permissions implemented for an object by the objects ACL Manager.
- Create and modify masks used to restrict allowable permissions.

Note: Standard UNIX tools that display and manipulate UNIX modes have an effect only on the ACLs established for the file system.

For a detailed description of the DCE control programs **acl** commands, see the *OS/390 DCE: Command Reference*.

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]
```

The example command replaces the ACL for the host **hermes** with the ACL for the host **cyclops** whose name is specified in the **acl show** command called by the **-acl** option. Note how the **-acl show** command in the **-acl** option is enclosed in brackets (**[ ]**). This is required when the **-acl** option value is a command invocation. To copy an **extended** entry type from the domain of one ACL Manager to the domain of another ACL Manager, use the output of **dcecp**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.

Generating ACLs from Files

A convenient way to create an ACL is to create and edit a text file so that it contains the desired ACL entries, and then generate the ACL from it by using an **acl replace** operation.

For example, assume the file **std_acl** contains the following entries:

```
{mask_obj:crwxid-}
{user_obj:crwxid-}
{group_obj:crwx---}
{other_obj:-r-----}
{user:lizt:crwx---}
{group:composers:-r-----}
{user:bach:crwx---}
{user:mozart:crwx---}
```

The following **acl replace** operation adds the entries in **std_acl** to an ACL named **/.../dresden.com/my_filesystem/opus**.

```
dcecp> acl replace /.../dresden.com/my_filesystem/opus -acl [cat std_acl]
dcecp>
```

The **acl replace** operation overwrites all ACL entries with the ones from the file **std_acl**. Regardless of what they were before, the ACLs for **opus** now look like this:

```
mask_obj:crwxid-
user_obj:crwxid-
user:lizt:crwx---
user:bach:crwx---
user:mozart:crwx---
group_obj:crwx---
group:composers:-r-----
other_obj:-r-----
```

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 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.

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. The directories are containers. The directories can hold simple objects (files) and other containers (subdirectories).

The Object ACL is associated with simple and container objects. The Initial Container and Initial Object ACLs are associated only with container objects.

Initial ACLs for Objects and Containers

Initial ACL entries and the ACL that contains them are applied automatically when an object is created. These ACLs can be changed at any time with the DCE control program (**dcecp**). The types of DCE ACLs used as initial ACL for containers and objects are as follows:

- The **Initial Container ACL** determines the default ACL for containers created within a container. For example, the file system Initial Container ACL for a directory specifies the default ACL for subdirectories created within that directory.
- The **Initial Object ACL** determines the default for objects created within a container. For example, the file system Initial Object ACL for a directory specifies the default 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. Figure 61 on page 322 illustrates how the default ACL is assigned to simple objects created in containers.

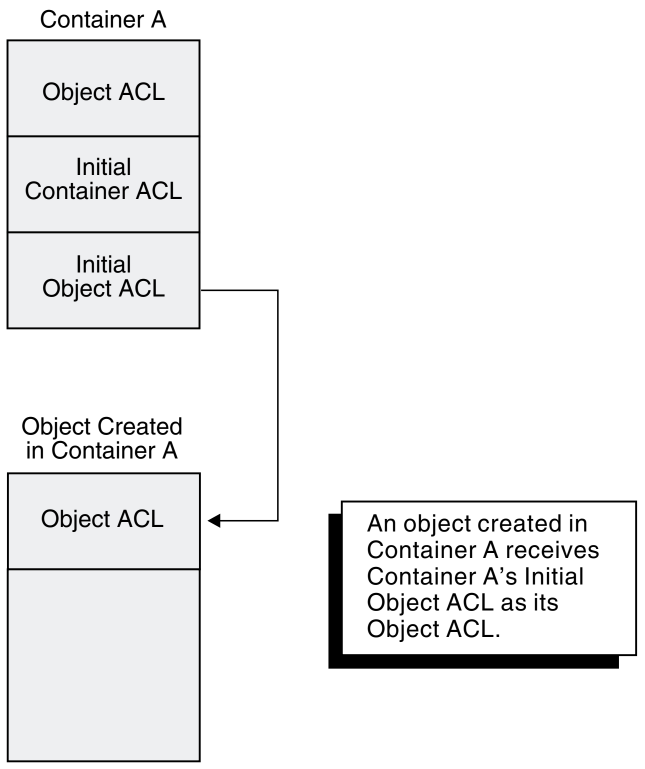

Container A
Object ACL
Initial Container ACL
Initial Object ACL
Object Created in Container A
Object ACL
An object created in Container A receives Container A's Initial Object ACL as its Object ACL.

Figure 61. 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 following ACLs of the parent container:

- Initial Container ACL as its Object ACL and as its Initial Container ACL
- Initial Object ACL as its Initial Object ACL.

For example, 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. Figure 62 on page 323 illustrates how the default ACLs are assigned to objects created in containers.

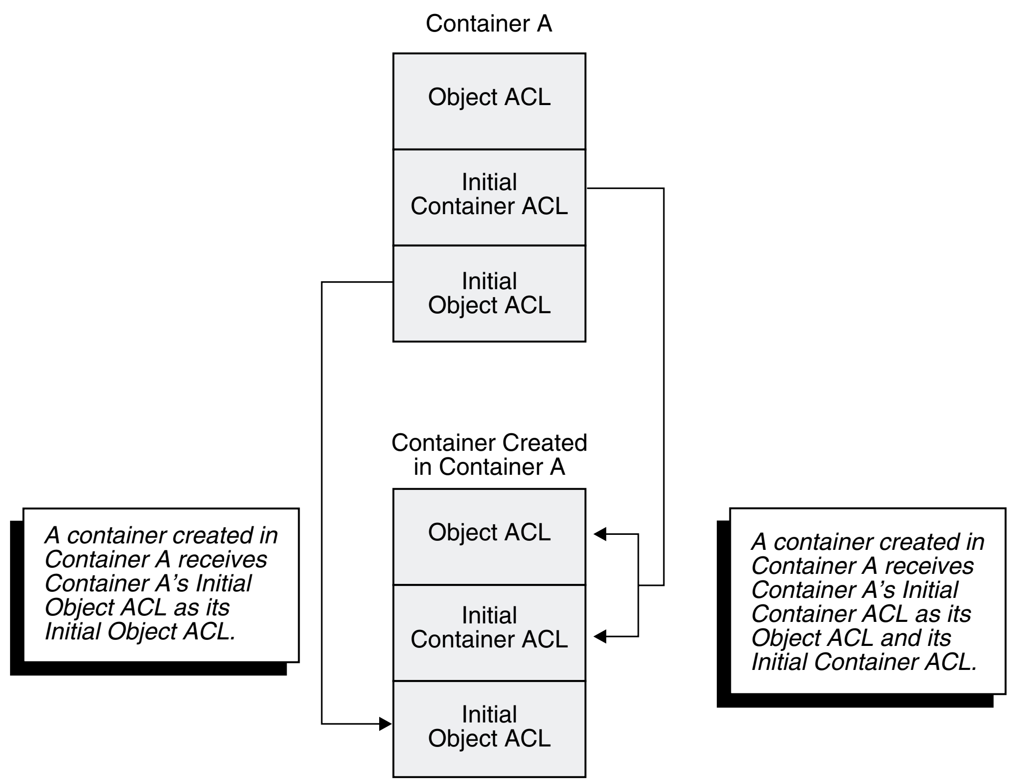

Container A
Object ACL
Initial Container ACL
Initial Object ACL
Container Created in Container A
Object ACL
Initial Container ACL
Initial Object ACL
A container created in Container A receives Container A’s Initial Object ACL as its Initial Object ACL.
A container created in Container A receives Container A’s Initial Container ACL as its Object ACL and its Initial Container ACL.

Figure 62. Initial ACLs for Containers Created in Containers

Default Container ACL Example: 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 (Initial Container ACL of Container A)

```
{user_obj crwxid}
{group_obj rw}
{other_obj r}
```

Initial Container ACL (Initial Container ACL of Container A)

```
{user_obj crwxid}
{group_obj rw}
{other_obj r}
```

Initial Object ACL (Initial Object ACL of Container A)

```
{user_obj crwxid}
{group_obj r}
{other_obj r}
```

Effect of Masks when Editing ACLs

If the user specifies a new **mask_obj** ACL entry, then **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.

The following example shows the way that mask recalculation works, and 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 users 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}
```

Chapter 35. 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.

This chapter provides information about the commands that the DCE control program (**dcecp**) offers for Security Service management. In addition, the chapter describes the commands that the **sec_admin** program supplies for reorganizing the Registry replica set in a cell.

Control program commands that you occasionally use for Security-related management tasks, such as **sec_create_db**, are not covered in this chapter. These are discussed in subsequent chapters of this guide along with the instructions for performing the tasks.

Using the DCE Control Program

Because detailed information about the DCE control program and its command syntax appears in Part 3, " The DCE Control Program" on page 43, this chapter does not repeat the information. It describes only the commands that the DCE control program provides specifically for managing the Security Service.

The DCE control program creates and maintains principals, groups, organizations, and accounts for the Security Services network-wide registry (Registry Service component). The control program also operates on the keytab files that protect the passwords for 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 **acl check** command displays the permissions that the ACL for a Security Service object grants to the invoking principal.

The following subsections describe the Security Service objects that the DCE control program operates on and the types of operations that the control program can perform on these objects.

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.
group	This object represents Registry groups. Groups are collections of principals for which you can assign access rights to objects.
organization	This object represents Registry organizations. Organizations are collections of principals to whom you can assign policies that expand your areas of administrative control.
account	This object represents the accounts that are established in the Registry for principals.
registry	This object represents the Registry (the Security Service database of account information) in a DCE cell. The Registry copy operated on can be either the master replica or a slave replica.

xattrschema	This object operates on the schemas (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 commands for managing the Security Service can perform the operations summarized in Table 16.

Table 16. Security Service Command Operations

Command	Operations
add	Adds a principal to a group or organization in the Registry replica.
catalog	Displays the names of all the principals, groups, and organizations in a Registry replica. For the Registry itself, displays the master and slave 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 slave replica. For an ERA schema, deletes entries. For a DCE ACL, removes ACL entries.
disable	Disables the master replica of the Registry for updates.
dump	Displays information on each replica of the Registry existing in a cell.
enable	Enables the master 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 a 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, 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 slave replica of the Registry to update its contents from the master replica.
verify	Checks if all of the Registry 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 Chapter 36, “Creating and Maintaining Principals, Groups, and Organizations” on page 331 and Chapter 37, “Creating and Maintaining Accounts” on page 347.

Using the Registry Editor

The following subsections explain how to start, stop, and get help for the Registry Editor. For detailed descriptions of all the Registry Editor commands, see the **rgy_edit** section in the *OS/390 DCE: Command Reference*.

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 run a sequence of Registry Editor commands. Most of the examples in this guide are in interactive mode. (See the **rgy_edit** section in the *OS/390 DCE: Command Reference* to use the Registry Editor in command-line mode.)

To start the Registry Editor in interactive mode, enter the following command:

```
$ dceshared/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=>
```

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
```

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
$
```

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
```

For detailed descriptions of all of the Registry Editor commands, see the **rgy_edt** section in the *OS/390 DCE: Command Reference*.

Using the sec_admin Program

While the DCE control program includes commands that create and maintain the master and slave 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 master and the slaves. 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 master replica, and the easiest way to move the master is to swap its role with a slave. The **sec_admin** program commands that you use to exchange master and slave replica roles are the **become** command and **change_master** command. The following subsections provide instructions for starting, stopping, and getting help on **sec_admin** and provide brief descriptions of the **become** and **change_master** commands.

Starting, Stopping, and Getting Help

To start the **sec_admin** program, enter the following command:

```
$ dceshared/bin/sec_admin
```

Once called, **sec_admin** enters the interactive mode in which you can run 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
```

The sec_admin Commands for Reconfiguring Replica Sets

Table 17 provides brief descriptions of the **sec_admin** commands for reconfiguring the Registry replica set. Instructions for changing master and slave replica roles can be found in “Changing the Master Replica Site” on page 423.

Table 17. The sec_admin Commands for Reconfiguring Replica Sets

Command	Operations
become_master	Causes a slave replica to assume the role of the master.
change_master	Causes the current master replica to give its role to a slave and to become a slave.

For detailed descriptions of the **sec_admin** commands, refer to the *OS/390 DCE: Command Reference*.

Chapter 36. Creating and Maintaining Principals, Groups, and Organizations

This chapter describes how to use **dcecp** to add, delete, and maintain principals, groups, and organizations. A **principal** can be defined as an entity that is capable of communicating securely with another entity. In other words, a principal is an entity that can be authenticated by the Security Service when it is associated with account information (that is, when **dcecp** is used to create an account for the principal). In actual practice, you can create principals to represent human users of the network, servers and machines in the network, and cells in the network with which you engage in cross-cell authentication. You can also create **groups** and **organizations**, which associate principals with certain access privileges and security policies. Creating DCE accounts, which is discussed in Chapter 37, “Creating and Maintaining Accounts” on page 347, incorporates the principal's association with groups and organizations.

This chapter begins with a discussion of the names that are assigned to principals, groups, and organizations and of the Universal Unique Identifiers (UUIDs) used internally by the Security service to identify registry objects.

Principal, Group, and Organization Names

You must assign a name to each principal, group, and organization in the registry. Although a principal, a group, and an 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, full, and aliases.

Primary Names

Primary names are assigned to principals, groups, and organizations. A registry object's primary name is the name 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, or organization itself.

Aliases

An **alias** is an optional alternative name for a primary name. Aliases can be assigned to principals and groups, but not to organizations. Aliases and the primary name for which they are an alternate share the same **Universal Unique Identifier** (UUID) and UNIX ID. UUIDs and UNIX IDs are described in “Universal Unique Identifiers and UNIX IDs” on page 333. 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 to establish several accounts for the same principal. For example, assume that for the primary name **mahler**, you create three aliases: **gustav**, **gus**, and **gm**. You can then create four 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 (for UNIX systems) and passwords and can be associated with different groups and organizations.

Because principals accrue only the privileges associated with the primary name or the alias that they use to log on, 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, 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 in to 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 number. See “Creating Aliases” on page 345 for information on creating aliases.

Name Formats

Names in the registry can contain any characters or digits, except the @ (at sign) and the **:** (colon)
| character. Principal names can contain slashes but *cannot* begin or end with a slash. They should not exceed 1014 characters in length.

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” on page 355 for details.)

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**

Object Creation Quotas

You can assign an object creation quota to each principal. This assignment lets you 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 no limits are placed on the number of objects the principal can create. A value of 0 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 zero, the principal is prohibited from creating registry objects unless you reset the object creation quota to a number other than zero using the **dcecp principal modify** command.

Note: 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 a principal's current object creation quota. This subcommand displays the total number of objects the principal is allowed to create at the current time (that is, the original quota minus the number of objects created by the principal).

Universal Unique Identifiers and UNIX IDs

The Security Service automatically associates a principal's, group's, or organization's primary name with a Universal Unique Identifier (UUID). UUIDs identify objects, 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.)

Usually, you do not have to be aware of UUIDs. They are created and maintained automatically. However, be aware that although the Security Service prints names and you can access objects by name, the Security Service identifies all objects internally by UUID. If you delete a principal from the registry, you also delete the principal's UUID. Any objects (files, programs) 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 using the DCE control program **principal create** command.) Therefore, you cannot simply add a new user and acquire a previously used UUID. You must use 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 in the command line for creating or modifying an account, you should avoid numbers under 100, because these are generally reserved for system accounts.

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. 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 Table 18 on page 334.

Table 18. Attribute Options to Create Principals

Option	Meaning
-fullname *namestring*	An optional name that more fully describes a primary name. To include spaces, enclose the full name in braces. The default is blank.
-uid *integer*	The required UNIX 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 UNIX number (the **maxuid** attribute) set with the **registry modify** command. However, you can enter a number lower than the low UNIX 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 low UNIX number and maximum UNIX 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 not quota. To prevent a principal from creating any registry objects enter a 0. The variable *quota* 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 incorrect logins. See “Extended Security Attributes for Principals” on page 336 for information on these “well-known” ERAs. See Chapter 38, “Creating and Using Extended Registry Attributes” on page 363 for information on ERAs in general.

Adding Principals

To add principals to the registry, use the **principal create** command. For example, 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 UNIX 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.

Note that it is possible to create multiple principals with one **principal create** command. To do so, enclose the principal names in braces, separated by spaces. For example, to create the principals **bach**, **britten**, **mahler**, and **satie**, you could enter:

```
dcecp> principal create {bach britten mahler satie}
dcecp>
```

If you create multiple principals, you must allow the principals UNIX 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. For example, the following sample command creates the principals **bach**, **britten**, **mahler** and assigns each an object creation quota of 5.

```
dcecp> principal create {bach britten mahler satie} -quota 5
dcecp>
```

If you wish to cross link the new DCE principal with an existing OS/390 RACF ID, you may want to use the RACF interoperability utility, **mvsimpt**, to create the DCE principal and account in the DCE registry instead of manually invoking the **dcecp** command. For more information, see “Cross Linking Existing RACF Users who are New DCE Users” on page 400.

Changing Principals

You can change a principal's primary name and any other information 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.

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>
```

Note that, if you change a primary name, that change is reflected in the membership lists of all the groups and organizations in 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 no longer be valid 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. Usually, 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. The following example illustrates these operations on remote host **persephone**.

```
dcecp> secval deactivate /.:/hosts/persephone
dcecp> secval activate /.:/hosts/persephone
dcecp>
```

Changing Principal Information:

Use the **dcecp principal modify** command to change any principal information except UNIX ID and User ID. 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>
```

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. If you delete the group alias **music_admin**, you also delete the **gustav music_admin classic** account.

Be aware that deleting a principal or alias could orphan the objects owned by the principal UUID.

The following example shows how to use the dcecp **principal delete** command to delete the principal named mahler:

```
dcecp> principal delete mahler
dcecp>
```

You can delete multiple principals or aliases with one **principal delete** command. To do so enclose the principal names in braces, separated by spaces. For example, to delete the principals **bach**, **britten**, and **mahler**, you would enter:

```
dcecp> principal delete {bach britten mahler}
dcecp>
```

Extended Security Attributes for Principals

You can attach 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 logins that are not valid
- Strength of principals' passwords as well as generation of passwords for principals
- Handling of login attempts by principals with expired passwords

These ERAs are introduced and explained in the following sections. See Chapter 38, “Creating and Using Extended Registry Attributes” on page 363 for information on how to use **dcecp** to attach these ERAs to principals.

OS/390 DCE Release 1 Authentication

With the addition of user preauthentication, OS/390 DCE Release 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 OS/390 DCE Release 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 plaintext 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 users 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.)

OS/390 DCE Release 1 reduces the likelihood of such attacks succeeding by providing for

- Preauthentication of principals making login requests (that is, by having the Security Service verify the identity of the requester before responding to the request)
- 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 will accept when authenticating the principal.

The preauthentication protocols are:

- The *third-party* protocol, which 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 list item.) OS/390 DCE Release 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 not running).
- The *timestamps* protocol, which provides an intermediate level of security. Timestamps preauthentication should be specified only for principals (such as cell administrators and non-interactive principals) who must be able to operate when the client is unable to construct a third-party authentication request as described above.

  In these cases, the client constructs and forwards a timestamps login request.

  Note: 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 1.0 (Kerberos V5) protocol, which authenticates pre-OS/390 DCE Release 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-OS/390 DCE Release 1 protocol.) This is the least secure level, and is provided only in order to enable OS/390 DCE Release 1 servers to accept login requests from pre-OS/390 DCE Release 1 clients. It is most vulnerable to the type of attack described above.

  Note: 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. Table 8 on page 199 provides a matrix describing the actions taken by the Authentication Service under the various combinations of login (authentication) request type and *pre_auth_req* ERA value.

For complete information on the details of DCE authentication (including the operation of the preauthentication protocols), see the chapter entitled “Authentication” in *OS/390 DCE: Application Development Guide: Core Components*.

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>
```

For further information on how to use **dcecp** to attach ERAs to principals, see Chapter 38, “Creating and Using Extended Registry Attributes” on page 363.

Preauthentication Interoperability Between DCE Versions: Table 19 shows how login requests are handled when OS/390 DCE Release 1 clients and servers interoperate with pre-OS/390 DCE Release 1 clients and servers in a single cell.

Table 19. DCE Release-to-Release Authentication Interoperation

Login Request Type	Pre-OS/390 DCE Server Response	OS/390 DCE Server Response
Pre-OS/390 DCE		
From pre-OS/390 DCE clients only.	No preauthentication. Returns pre-OS/390 DCE (unpreauthenticated) response.	Supports preauthentication. Checks for *pre_auth_req* ERA instance. If no ERA exists, or existing ERA has value=0 (NONE), returns MVS/ESA OpenEdition DCE Release 1 (unpreauthenticated) response. Otherwise, rejects login request.
TIMESTAMPS		
From OS/390 DCE clients only.	No preauthentication. Server ignores preauthentication data in request and returns pre-OS/390 DCE (unpreauthenticated) response.	Supports 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 OS/390 DCE TIMESTAMPS response. If existing ERA has value=2 (PADATA-ENC-THIRD-PARTY), rejects login request.
THIRD-PARTY		
From OS/390 DCE clients only.	No preauthentication. Server ignores preauthentication data in request and returns pre-OS/390 DCE (unpreauthenticated) response.	Supports preauthentication. Returns OS/390 DCE THIRD-PARTY response.

Managing Logins that are Not Valid

When you specify a preauthentication level of 2 (PADATA-ENC-THIRD-PARTY) for a principal, the OS/390 Security Server is able to detect and track incorrect login attempts for that 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 incorrect login attempts before the principal's account is disabled. (A successful login resets the counter.)
- Specifying the period of time the principal's account will be disabled after 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 incorrect login attempts the OS/390 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.

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>
```

Note: With OS/390 DCE Release 1, the incorrect login handling functionality accurately tracks login activity in a cell with 1 master and no replicas, but does not keep accurate counts in replicated cells. This is because:

1. Login attempts in a replicated cell are randomly assigned to either a master or replica.
2. 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.

For further information on how to use **dcecp** to attach ERAs to principals, see Chapter 38, “Creating and Using Extended Registry Attributes” on page 363.

Managing Password Strength and Password Generation

The DCE password format policy is described in Chapter 42, “Maintaining Policies and Properties” on page 411. This 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

You can extend these password strength policies in your cell by creating a password management server to perform customized password checking and generation. DCE provides a password validation and generation server, **pwd_strengthd**, which you can use as the basis for a password management server.

DCE also provides a Password Management API which application developers can use to acquire information about the principals password management policy, and to request generated passwords from the password management server. See the chapter entitled “The Password Management Application Programming Interfaces” in *OS/390 DCE: 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 attaching instances of the *pwd_val_type* and *pwd_mgmt_binding* ERAs to the principal as follows:

pwd_val_type
: Specify password creation options for the principal as follows:

  0 **(NONE)** Specifies that the principal's password is subject only to DCE standard policy. (See Chapter 42, "Maintaining Policies and Properties" on page 411 for a description of 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 principals can either supply password text, or indicate that they want the password management server specified in the *pwd_mgmt_binding* ERA to generate a password for them.

  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 cells password management server.

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} {/.:/subsys/dce/pwd_mgmt/pwd_strength}}}}
dcecp>
```

Note: The protection level is **pktprivacy**, which is only available if User Data Privacy (DES and CDMF) feature is installed. See "Specifying the Authentication Type" on page 369 for more information.

For further information on how to use **dcecp** to attach ERAs to principals, see Chapter 38, "Creating and Using Extended Registry Attributes" on page 363. For information on requesting generated passwords when changing a password, see "Generating Passwords Using dcecp."

For information on configuring a password management server, see "Managing a Password Management Server."

Managing a Password Management Server: The section on configuring DCE in *OS/390 Program Directory*. explains how to use the DCECONF command to configure a password management server. This section provides additional notes on managing a password management server.

- To protect password security, and to optimize performance, the password management server should run on the same machine as the master DCE security server.
- While the DCECONF command supports configuration of only one password management server in a cell, it is possible to manually configure additional servers. Principal *pwd_mgmt_binding* ERAs can then be set to point to the appropriate server for each principal.

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 to this:

```
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 -mypwd -dce-
```

Note: If you are in the TSO environment, a blank must follow the password (**-dce-**) before you press Enter.

Attention

Never run the following **dcecp** command, because the password will be changed in the account, but the user will not see the new password:

```
dcecp> account mod smitty -password [account gen smitty] -mypwd -dce-
```

Adding and Maintaining Groups and Organizations

A group or organization must have been added to the registry before it can be used in association with an account. When you add groups, using the **dcecp group create** 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.)

Project Lists

A principal's **project list** is a list of all the groups in which a principal or alias is a member.

The principal's access privileges to an object are contained in the Access Control List (ACL) of the object. (See Chapter 34, “Using Access Control Lists” on page 307 for a description of ACLs.) In the ACL, access privileges can be explicitly given to the principal by assigning certain privileges to the principal name. However, in the absence of this, access privileges can also be implicitly accrued by the principal through the privileges that are given to the groups of which the principal is a member. That is, the principal has the access privileges that accrue from membership in every group named in the object's ACL. For example, assume the ACL for file X contains two entries: one permits group A write access and one permits group B read access. Then, any principal who is a member of both groups A and B can read and write to file X.

Project Lists and Rights: A principal accrues privileges through the project list that is associated with the name it uses to log in to DCE. (It does not accrue rights from its names nor any of its aliases.) Note that the principal's primary and alias names are associated with distinct (and probably different) project lists. A principal can have aliases with different accrued privileges. For example, assume a principal named **gustav** is a member of groups A and B. Under the alias **gus**, **gustav** is also a member of groups C and D. When the principal **gustav** logs in, he accrues access privileges from groups A and B only. He accrues access privileges from groups C and D only when he logs in with the alias **gus**.

To display the groups in which a principal (or its alias) is a member, use the **principal show** command described in “Displaying Principal Information” on page 384

Prohibiting Inclusion on Project Lists: If a group is prohibited from inclusion in a project list, its privileges 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 permissions, not write permissions. 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 can decide to prohibit some groups from inclusion on the list. You may, for example, want to prohibit any reserved groups with access privileges 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 to the registry. 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 in Table 20.

Note that when you use the **dcecp group create** command and **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” on page 334 for details.

Table 20. Attribute Options to Create Groups and Organizations

Option	Meaning
-gid	The required UNIX 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 UNIX number (the **maxuid** attribute) set with the **dcecp registry modify** command. However, you can enter a number lower than the low UNIX 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 low UNIX number and the maximum UNIX number.
-fullname *string*	An optional name that more fully describes 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. **no** is the default.

Example: Adding a Group: The following example shows how to add a group named **symphonists** to the registry.

```
dcecp> group create symphonists
dcecp>
```

In the example, the group UNIX ID is generated automatically, no full name is supplied, and the group is included on project lists.

Example: Adding an Organization: The following example shows how to add an organization named **classic** to the registry.

```
dcecp> organization create classic
dcecp>
```

In the example, the organization UNIX ID is generated automatically and no full name is supplied.

Changing Groups and Organizations

For groups and organizations, you can change the primary name and full name. In addition, for groups, you can change whether the group can appear in project lists, and for organizations, you can change the policy. (See Chapter 42, "Maintaining Policies and Properties" on page 411 for details on setting and changing organization policy.)

Use the **dcecp group modify** command to change groups. The following example shows the use of this command with the **-inprojlist** option to change the group **symphonists** project list inclusion property from **yes** (include on project lists) to **no** (prohibit from project lists).

```
dcecp> group modify symphonists -inprojlist no
dcecp>
```

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:

```
dcecp> group rename old_name -to new_name
dcecp> 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.

The following example shows the **group rename** command used to change a full name from **symphonists** to **symphonists7**.

```
dcecp> group rename symphonists -to symphonists7
dcecp>
```

Note that, if you change a primary name, that change is reflected in the membership lists of all the 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. The following example shows the **group delete** command being used to delete the group **symphonists**.

```
dcecp> group delete symphonists
dcecp>
```

The next example shows the **organization delete** command being used to delete the organization **classic**.

```
dcecp> organization delete classic
dcecp>
```

Note that 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.

Maintaining Membership Lists

Each group or organization has a **membership list**, listing the principals who are members of the group or organization. Use the **dcecp group add** command to add members to the membership list, 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 that deleting the principal from a group or organization can affect the principal's access privileges to objects. This change takes effect only when the principal's **ticket-granting ticket** is renewed. See Chapter 37, "Creating and Maintaining Accounts" on page 347 for more information on ticket renewals.

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 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, tries to add the principal to the group and organization. For example, assume 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 permissions to the group, the command fails and displays a message like the following:

```
Not authorized to perform operation
```

Example: Adding and Deleting Group Members

The following example shows the use of the **dcecp group add** command with the **-member** option to add **mahler** to the group **symphonists** and delete **strauss** from the group **symphonists**

```
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 braces ({ }) separated by spaces. For example to add the principals **bach**, **britten**, and **mahler** to the group **symphonists**, you would enter:

```
dcecp>group add symphonists -member {bach britten mahler}
```

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 no longer valid 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. Usually, 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. The following example illustrates these operations on remote host **persephone**.

```
dcecp> secval deactivate /.:/hosts/persephone
dcecp> secval activate /.:/hosts/persephone
dcecp>
```

Creating and Maintaining Aliases for Principals or Groups

Use the **dcecp principal create** command to create and maintain aliases for principals and groups. Organizations cannot be given 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 that *name* is an alias. Its value is YES or NO.

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. Its value is YES or NO.

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.

Chapter 37. 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 in tandem, but independently of each other. A principal's network identity is defined by an account in the network registry. A principal's local identity (in OS/390 DCE) is defined in the security subsystem (such as RACF).

Registry accounts define a network identity by associating a principal with a group, an organization, and related account information, such as the password that authenticates 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
- Any cell with which you engage in authenticated cross-cell communication. (Accounts for cross-cell authentication are special types of accounts that are described in Chapter 39, “Administering a Multicell Environment” on page 375).

This chapter describes:

- Each type of account and how to create and maintain it
- How accounts are authenticated and how to display privilege attributes and tickets
- How to create and maintain the keytab file that stores keys for server principals

User Accounts

User accounts are associated with the user's password and information that is used when the user logs into DCE. Account information includes such things as the principal's home directory and login shell, and authentication policy, which define parameters that help control a principal's access to DCE. Use the **dcecp account create** command to create accounts for human users, the **dcecp account modify** command to change them, and the **dcecp account delete** command to delete them.

Server Accounts

Servers, which can also be called **applications**, that engage in communication 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 a login and authenticate that identity. Therefore, you must use the **dcecp account create** command to create registry accounts for these servers.

Passwords for Server Accounts

During login, all principals (human, server, and machine) must provide 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 (based on APIs supplied with DCE), is to store server keys (passwords) 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 the principal's local identity can read or write to the file.

You can access the keytab files 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 running under the identity of the local machine, a separate keytab file needs to be used for each server. During login, the server can access this file to obtain its key, pass its key to the Authentication Service, log in, and be authenticated.

Use the **dcecp keytab add** command to add keys for servers to a keytab file and the **dcecp keytab remove** command to delete server keys.

If OS/390 DCE Daemons' Passwords Expire: The DCE daemons usually change their passwords automatically before the passwords expire. However, this can only happen if the daemons are running. If the password expires while the daemon is not running, that daemon cannot be restarted.

In this case, you will have to perform the following:

1. Use **dcecp** to change the password of the OS/390 DCE daemons in the Security registry.
2. Use the **dcecp keytab add** command to enter the new password in the keytab file. This command is described in the next section.

Accounts for Servers

Accounts for servers are created using the **dcecp account create** command in the same manner as creating accounts for users. The only difference is that after you run **dcecp** you must run the **keytab add** command on the machine on which the server resides to add the server's password to the **/krb5/v5srvtab keytab** file. The password you enter in the keytab file (using the **keytab add** command) must match the password you entered when you created the server's account. You can manually ensure that these passwords are the same or you can specify that the **keytab add** command sets the server's registry password at the same time that it sets the server's password in the keytab file.

Steps for Creating 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 an entry to the keytab file. The server's password in the registry and the server's key in the keytab file must match. You can ensure that these passwords are the same by manually entering the same passwords in both commands, 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.

Machine Accounts

All machines must also 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 Part 6, "DCE Directory Service" on page 157 for more information on names in the cell namespace.

How Identities Represented by Accounts Are Authenticated

When a principal logs in to DCE, the security client uses the password the principal supplies (or that is supplied for it in the case of a server or machine principal) to derive the principal's authentication key. A copy of the principal's authentication key exists also in the registry database, having been stored there when the principal's account was created (or when the password was changed.) It is thus 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.
   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.
   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, then the password supplied by the principal, and used by the security client to derive its version of the principal's authentication key, is not valid (that is, different from the password used to derive the registry's copy of the principal's authentication key), and login is denied.

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 when principals request access to objects to determine their privileges to those 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 a principal's privilege attributes are authenticated. (The DCE Access Control Lists, used to control access to DCE objects based on a principal's privilege attributes, are described in Chapter 34, "Using Access Control Lists" on page 307.)

Ticket-Granting Tickets and Tickets to Services

A **ticket-granting ticket** (TGT) 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**. They tell the server to consider the principal an **authenticated user** for access checking because the Security Service guarantees the principal's identity to be as stated in the principal's privilege attributes.

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 unauthenticated 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 logging in to DCE again.

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 *OS/390 DCE: 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 principal 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 times 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. Before this happens, the principal should change the password using 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.

The **kdestroy** command is described in the *OS/390 DCE: Command Reference*.

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 a UNIX **/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.

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” on page 414 for more information on contradictory policy.)

Table 21 on page 352 lists 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.

Table 21 (Page 1 of 2). Attribute Options to Create Accounts

Option	Meaning
-acctvalid {yes|no}	A flag that determines account validity. If you set this flag to **no**, the account is not valid and the account principal cannot log in to 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) on which the account expires. To renew a 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 for which tickets were obtained fraudulently before the password was changed. To eliminate the principal's access to the system, the tickets must be canceled. If you set the **-goodsince** attribute to the date and time the compromised password was changed, all tickets issued before that time become invalid 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. 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 plaintext. The system encrypts the password you supply. 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 will be prompted to change it the next time that the principal logs into the account. The default is **yes**.

Table 21 (Page 2 of 2). Attribute Options to Create Accounts

Option	Meaning
-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 is run 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. (See "Authentication Policy" on page 413.)
-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, because 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. (See "Authentication Policy" on page 413.)

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 usually 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.

Adding Accounts Example

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.

Because you are required to enter your password, you must run the **account create** 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 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 -password password
dcecp>
```

Note that 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: The back slash (\) is the line continuation character.

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.

Modifying Accounts

The **dcecp account modify** command lets you 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. If you do not enter a password, you are prompted for it. Because you are required to enter your password, you must run the **account modify** 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>
```

Reactivating Accounts that have Expired

Use the **login_activity** command to reactivate a user account that has expired. This command displays or resets the login activity information for a DCE principal. With this command, you can do the following operations on a specified principal:

- Query the principal's login activity information.
- Reset the number of unsuccessful login attempts the principal is allowed.
- Reset the principal's disable time attribute.

The login activity information that is capable of being reset is defined in extended registry attributes (ERAs).

Currently, attributes can only be set on a DCE principal. The **login_activity** command allows the DCE administrator the capability of overriding these values by resetting the login activity information for a principal.

Note: Although this is intended for use as an administrator command, the **login_activity** command can also be used by a DCE end user.

In order to reactivate an account that has expired, set the principal's disable time to **0**, as in this example for principal **judith**:

```
login_activity reset -p judith -d 0
```

This immediately enables **judith**'s account.

Here are some other examples of how to use the **login_activity** command:

1. To query the login activity information for DCE principal **jonathan**, enter:

   ```
   login_activity query -p jonathan

   Login activity information:
            Principal name: jonathan
            Current number of unsuccessful attempts: 0
            Account disabled until: <not disabled>
            Last successful login: Wed Sep  6 17:18:06 1996 from 9.130.79.38[1175]
            Last unsuccessful login: <none> from <none>
   ```

   In this example, the **login_activity** *query* command displays the login activity information for the principal **jonathan**. This information includes the current number of unsuccessful login attempts, the time until which the principal is disabled, and information regarding the last unsuccessful and successful login attempts.

2. To reset the disable time for the DCE principal **jordan**, enter:

   ```
   login_activity reset -p jordan -d 96/10/16.08:52

   login_activity query -p jordan

   Login activity information:
            Principal name: jordan
            Current number of unsuccessful attempts: 1
            Account disabled until: Mon Oct 16 08:52:00 1996
            Last successful login: Mon Oct  9 09:32:29 1996 from 9.130.79.38[1141]
            Last unsuccessful login: Wed Oct 11 16:59:35 1996 from 9.130.79.38[1175]
   ```

   In this example, the **login_activity** *reset* command changes the time until which the DCE principal **jordan** remains disabled to 8:52am on October 16, 1996.

For the complete syntax and usage of the **login_activity** command, see the *OS/390 DCE: Command Reference*.

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.

The following subsections describe how to manage keytab files.

The Keytab File

Keytab 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 lets you 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 pathname 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.
keytab_name	Is the name of the keytab object.

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 lets you 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.

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.

Table 22 lists the other options you can supply to the **keytab create** and **add** commands.

Table 22 (Page 1 of 2). The keytab create and add Options

Option	Meaning
-local	This option lets you access the keytab file without using **dced**.
-noprivacy	This option lets you 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.

Table 22 (Page 2 of 2). The keytab create and add Options

Option	Meaning
-storage *local_file_name*	The pathname 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 pathname of the **dced** object to be created as an argument, the **storage** option to specify the keytab's local file name, the **data** option to specify the name of the server principal and the keys, and any of the appropriate options listed in Table 22 on page 358.

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 a 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}}}
```

Notes:

1. The back slash (\) is the line continuation character.
2. The keytab file created by this command using the **-storage** attribute is owned by a root user ID in read/write mode. On OS/390, the root user ID STC1 is used.

   To get access to this keytab file when a server is started:

   – The server must have root authority, or

– The permissions of the file must be changed to give access to the server's local identity. For example, if the server is going to run from the GSERV1 user ID, the keytab file should be updated so that its owner is GSERV1.

Adding Entries to a Keytab File: Use the **keytab add** command to add entries to an existing keyfile. When you use this command, you must supply the name of the keytab file's **dced** object and any of options described in listed in Table 22 on page 358.

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
```

Note: The **-random -registry** option pair works only if the principal designated with the **-member** option has a valid password in the keytab file.

Removing Entries from Keytab Files: 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.

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 -member svr_2 -type des
dcecp>
```

Removing Keytab Files

You can remove local 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. For example, 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>
```

To delete only the **dced** object, use the **-entry** option.

For example, 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>
```

Changing Server and Machine Passwords in the Keytab File

Passwords for all principals must be changed when they expire. The **dced** daemon's Security Validation Service automatically changes the machines 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**, as described in “Creating a Keytab File” on page 359.

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. Because the human principal can never know the randomly generated password, the principal cannot log into the system and cannot reauthenticate.

Handling Compromised Server or Machine Passwords in the Keytab File

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.

Chapter 38. Creating and Using Extended Registry Attributes

The Registry stores specific information about principals, groups, organizations, and accounts. This is the information that you create when you use the **dcecp** command to create principals, groups, organizations, and accounts. The kind of information that can be stored in the registry database is defined in the registry schema, which is essentially a catalog of the kinds of data stored in the database. There is a schema entry definition for each type of attribute that can be associated or attached to a registry object. For example, a schema entry defines principal names as a printable character string in PCS (portable character set) format. When you create a principal, you enter a text string that is stored in PCS format.

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 schema. After the extended attribute types are defined, 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 are passed to those applications for processing. For example, if you work with an OS/390 application that requires a user's OS/390 name, you could establish an OS/390 name extended attribute that is stored in the registry. The OS/390 name can then be passed to the OS/390 application for appropriate processing.

If a principal has extended attributes, these attributes are 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 1) is attached to a registry object and 2) has a value.

This chapter describes the how to create and maintain attribute types and attribute instances. It begins first with a discussion of the **xattrschema** object; then it describes how to define attribute types and attach attributes to objects.

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.

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.

Creating Attribute Types

Use the **dcecp xattrschema create** command to create attribute types. When you use this command you can supply the attribute options summarized in Table 23 on page 364. Note that the options described in this table can also be supplied without the hyphens in attribute lists.

Table 23. Options to Create Extended Attributes

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. No default. Attribute type ACL Managers are described fully in "Defining the ACL Managers for Attributes" on page 366.
-annotation *string*	A PCS text string that annotates the attribute type. If the string contains spaces, enclose it in braces or quotation marks. Default is blank.
-applydefs	This option is not currently implemented.
-encoding *type*	The format of the attribute type instance value. Attribute encoding is described more fully in "Defining the Attribute Type Encoding" on page 367.
-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 coded 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*	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 368 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 type A is attached to 25 principals in the cell. If the attribute type A is coded as unique, the value of the A attribute 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*	A UUID that identifies the attribute type 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 will generate one.

Use the **dcecp xattrschema create** command to create attribute types. The syntax of this command is as follows:

xattrschema create *attr_name* **{***attr_options***}**

attr_name	Is the fully qualified name of the attribute type to create.
attr_option	Is one or more of the options described in Table 23.

The following sample command creates the extended attribute type named **employee_num** and assigns it an an ACL Manager of **principal** and an encoding type of **integer**.

```
dcecp> xattrschema create /.:/sec/xattrschema/employee_num \
-aclmgr {principal r r r D} -encoding integer
dcecp>
```

Although the sample above uses options to supply information, you can use standard **dcecp** attribute lists.

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.

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.

The following sample 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>
```

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.

The following example shows the **xattrschema rename** command used to change an attribute name from **log_name** to **MVSname**.

```
dcecp> xattrschema rename /.:/sec/xattrschema/log_name -to MVSname
dcecp>
```

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
```

where *attribute_name* is the fully qualified name of the attribute to be deleted.

For example to delete the extended attribute named **MVSname**, the command would be:

```
dcecp> xattrschema delete /.:/xattrschema/MVSname
dcecp>
```

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 do 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 DCE:

policy	To access the ACL manager for the policy object.
principal	To access the ACL manager for principals.
group	To access the ACL manager for groups.
organization	To access the ACL manager for organizations.
secdirectory	To access the ACL manager for directories in the registry database.
replist	To access the ACL manager for the replica list.
xattrschema	To access 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).

For example, 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-82d4-08000925135” 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

Note that 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 an attribute's encoding. This option has the form:

```
-encoding type
```

The *type* parameter is one of the encoding types described in Table 24.

Table 24 (Page 1 of 2). Encoding Types

Encoding Type	Meaning
any	Not implemented in this release of the 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 allow for 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.
confidential	Not implemented in this release of the DCE.

Table 24 (Page 2 of 2). Encoding Types

Encoding Type	**Meaning**
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.
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 simply a marker that is either present or absent.

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.

Trigger servers are called 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 supported starting with OS/390 DCE Release 1.

To associate an attribute type with a trigger server, use the **-trigtype** and **-trigbind dcecp xattrschema** options.

The trigtype Option

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 supported starting with OS/390 DCE Release 1.

Once 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.

cdmf
: Protects as specified by all the previous levels and also encrypts each RPC argument value.

This level encrypts all user data in each cell and provides a lower level of packet privacy than **pktprivacy**. Although this is the second-highest protection level, it is available only if User Data

Privacy (DES and CDMF) feature or User Data Privacy (CDMF) feature was installed.

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. This protection level is only available with DES, so it requires that User Data Privacy (DES and CDMF) feature be installed.

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. 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:

```
ncadg_udp_ip 130.105.96.3 1234
```

Sample Value for the trigbind Option: 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, uses and authorization service of DCE shared secret key and a 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}}
```

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 just 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

  Note: In the current release, you cannot attach attributes to the policy object.
- Directory
- Replist
- xattrschema

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.

For example 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>
```

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. For example, 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>
```

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 replace by the new values specified in the command. For example, to change only a specific value, you must enter all the values. For example 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>
```

However, you can add and remove individual values from a multivalued attribute. Use the **-add** option to add values. For example, 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>
```

(Use the **remove** option described in the following subsection to delete specific values in a multivalued attribute.)

Note that 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>
```

If the **multi_name** attribute had the following values:

```
{multi_name value1 value2 value3}
```

The command above 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. For example the following command deletes all instances of the **MVSname** attribute from the principal named **delores**.

```
dcecp> principal modify delores -remove MVSname
dcecp>
```

To remove a single instance of a multivalued attribute, supply the attribute name and the attribute value. For example 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>
```

However, if you delete the last instance of a multivalued attribute, **dcecp** will also delete the attribute from the object because an attribute without a value cannot be attached to an object.

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. For example 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** assumes that **MVSinteger** is the value for the **MVSname** attribute and, because no such value exists, the command will not succeed.

Using Attribute Sets

At 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 attribute 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. Because 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 multivalued attributes. The attribute instances that are members of the set are identified by UUIDs.

Chapter 39. Administering a Multicell Environment

Previous chapters in this guide described 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 **.../sec/principal/krbtgt** directory, (which contains cell principals) in the registry database to allow updating only by 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.) Note that, 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 when the registry is created. (See Chapter 46, “Accessing Registry Objects” on page 429 for information on the structure of the registry database and setting ACLs for objects in the registry database.)

This chapter describes the trust relationships between cells that allow principals from foreign cells access to objects in your cell and principals from your cell access to objects in foreign cells.

Trust Relationships

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 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.

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 in to accounts in the local cell and users in the local cell to log in to accounts in the foreign cell.

Direct Trust Relationships

In a direct trust relationship, two cell's 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 to be authenticated to the other cell. A direct trust relationship involves only two cells.

Establishing Trust Relationships

Use the **dcecp registry connect** command to establish a direct trust relationship. 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 the reverse. You control foreign principals' access to specific objects with ACL entries, just as you do for principals in the local cell.

When the accounts are created, the **dcecp registry connect** command performs two tasks that you should be aware of. First, it automatically generates *one* 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 a UNIX number that is shared by all principals that are in a given 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.

Creating Trust Relationships

To create peer-to-peer relationships, follow these steps:

1. Ensure that the two cells are enabled to communicate. See “Enabling Other Cells to Find Your Cell” on page 254 for more information.
2. Run the **dcecp 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).
3. Optionally, use the **dcecp account modify** command to fine tune the attributes of the account (assigned by default when the account was created). For example, the account's expiration date (**expdate** attribute) defaults to **none**. You may want to enter a date to ensure that the account will be actively renewed after a period of time.
4. 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 not valid, no cross-cell communications can take place.

Command Options for the registry connect Command

When you use the **dcecp registry connect** command, you must supply the fully-qualified name of the foreign cell with which you will establish a peer-to-peer relationship. This name is stripped off the full pathname, 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 **dcecp registry connect** uses your local cell name for the primary name of the local cell's account principal. This name is stripped off the full pathname and prefixed with **krbtgt**, just as the foreign cell name is.

Table 25 lists additional information that you can supply to the **registry connect** command.

Table 25. The registry connect Command Options

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 you with 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 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 those necessary to complete the 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, an error message is displayed. If **none** is specified, the group does not have to be predefined.
-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 group does not exist, an error message is displayed. If **none** is specified, the organization does not have to be predefined.
-expdate	The time and date that both the local and the foreign cell's accounts expire, and the peer-to-peer relationship is ended, prohibiting any further authenticated communications between principals in the two cells. To renew the accounts, change the date in this field. The default is **none**.

Example: Creating Cross-Cell Authentication Accounts: The following example of a **dcecp registry connect** command creates an account for the foreign cell identified by **/.../dresden.com**. The local account is associated with the group named cell_group_local and the organization named cell_group_dres and the 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 none -fgroup none \
                             -org none -forg none \
                             -mypwd cell_admin
dcecp>
```

Note that the back slash (\) is the line continuation character.

The Accounts Created by the registry connect Command

The accounts and principals that are created by the **dcecp registry connect** command are given default attribute values listed in Table 26. 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.

Table 26. Default Attribute Values of Cross-Cell Authorization Principals and Accounts

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 pathname and prefixed with **krbtgt**.
fullname	The cell's pathname.
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, for example, that, 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 from 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.

Changing Cross-Cell Authentication Accounts

You can change the account created by the **dcecp 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** (not valid). For standard accounts, setting the attribute to **no** causes users to be prompted 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 shared by Authentication Services, you must rerun the **dcecp registry connect** command to recreate the accounts and create the properly synchronized passwords.

Generally, do not delete the accounts or the account's principals unless you are breaking the peer-to-peer relationship with the cell. If one of the accounts is deleted, you must run the **dcecp registry connect** command to recreate both accounts and restore the peer-to-peer relationship.

Chapter 40. Viewing Registry Information

Using the **dcecp**, 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 keytable.
- The **show** command displays information about a specific instance of an object.

This chapter describes how to display operation available for all Security objects, except the registry object which is described in Chapter 43, "Performing Routine Maintenance" on page 419.

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.

To display all accounts in the registry database with names prefixed by cellname, enter:

```
account catalog
```

To display all accounts in the registry database with names *not* prefixed by cellname, 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 -policies
```

To display all attributes and all policies for a named principal's account, enter:

```
account show acct_name -all
```

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 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.

To display all groups or organizations in the registry database with names prefixed by cellname, enter:

```
group catalog
```

or

```
organization catalog
```

To display all groups or organizations in the registry database with names *not* prefixed by cellname, enter:

```
group catalog -simplename
```

or

```
organization catalog -simplename
```

To display all members of a specified group or organization with names prefixed by cellname, enter:

```
group list group_name
```

or

```
organization list org_name
```

To display all members of a specified group or organization with names *not* prefixed by cellname, enter:

```
group list group_name -simplename
```

or

```
organization list org_name -simplename
```

To display all attributes for a group or organization, enter:

```
group show group_name
```

or

```
organization show org_name
```

To display all extended attribute instances attached to a group or organization, enter:

```
group show group_name -xattrs
```

or

```
organization show org_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 org_name -all
```

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 shows 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 where it says "{alias no}." This line indicates that the name **users_temporary** is the primary name, not an alias name. For an alias, this line would read as "{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 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.

To display all principals in the registry database with names prefixed by cellname, enter:

```
principal catalog
```

To display all principals in the registry database with names *not* prefixed by cellname, 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**

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

Use the **dcecp xattrschema catalog** and **xattrschema show** commands to display information about the extended attribute types. Note that 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 database, this name is **/.:/sec/xattrschema**. 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.

To display the names of all attribute types in the registry database with names prefixed by cellname, enter:

xattrschema catalog /.:/sec/xattrschema

To display all attribute types in the registry database *not* prefixed by cellname, enter:

```
xattrschema catalog /.:/sec/xattrschema -simplename
```

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
```

The following example, lists the names of all extended attributes in the registry prefixed by cellname:

```
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/test_integer
dcecp>
```

The following example, list 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>
```

Displaying ACL Information

Use the **dcecp acl show** commands to display a ACL entries 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.

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
```

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 keytab Information

Use the **dcecp keytab catalog**, **keytab list**, and **keytab show** commands to display information about key tables that manage server passwords on DCE hosts. 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.

To display the names of all keytab files on a specified host with names prefixed by cellname, enter:

keytab catalog *host_name*

where *host_name* is of the form **/.../cell/hosts/hostname**. 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 cellname, 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*

where *file_name* is of the form **/.../cell/hosts/hostname/config/keytab/name**. To display the attribute list of the key table in the named keytab file, enter:

keytab show *file_name*

The information displayed includes the **dced** object's UUID and annotation string, the keytab's local file name, and the principal list with associated key type and version numbers.

To display the local names of a specified key file, enter:

keytab show *file_name* **-entry**

To display all entries in a key file, including the keys, enter:

keytab show *file_name* **-members**

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}
{mahler des 2}
dcecp>
```

Chapter 41. RACF Interoperability and Single Sign-on

OS/390 DCE allows interoperability between Resource Access Control Facility (RACF) on OS/390 and DCE. This security interoperability allows a DCE client to access a DCE-enabled server on an OS/390 system and allows the DCE-enabled server to acquire corresponding local security credentials for the DCE client for access to OS/390 resources. The interoperability function provides:

- Information necessary for DCE-enabled servers running on OS/390 to use OS/390 user IDs on behalf of their DCE clients. Appropriately authorized DCE servers can also log the DCE user on to OS/390 (acquire OS/390-RACF credentials) if needed.

  This part of interoperability requires that a DCE server obtain the OS/390 user ID of the DCE principal whose client application called the server. This information is contained in a new RACF general resource class, DCEUUIDS. DCEUUIDS contains a DCE principal's UUID and associated OS/390 user ID.

- Information necessary to transparently log an OS/390 user in to DCE when necessary, without prompting for a DCE user ID or password. This ability is called **single sign-on**.

  Single sign-on requires that the DCE principal's name and password be available to the OS/390 system when it needs to log a user in to DCE. This is done with information from a new RACF DCE segment associated with an OS/390 user ID. The segment information allows the OS/390 user to authenticate to OS/390 and run a DCE program without reauthenticating to DCE. For more about single sign-on, see “Single Sign-on for OS/390 and DCE” on page 409.

- Administration commands provided by RACF and utilities provided by OS/390 DCE to populate the RACF database with cross-linking information. There are also Security Authorization Facility (SAF), UNIX System Services, and C language application programming interfaces (APIs) to access this information. Getting the cross-linking information into (and out of) the RACF database is what allows interoperability and single sign-on to work.

Notes:

1. Although the discussion in this chapter focuses on RACF, any OS/390 external security manager (ESM) that has equivalent support can be used instead of RACF. However, OS/390 DCE provides utilities for cross linking information only between DCE and RACF. If you are not using RACF as your ESM, see the publications that come with your ESM product to determine if similar utilities are provided with the product.
2. Before you start any DCE server, be sure that the OS/390 user ID under which it will be started has either of the following:

   - No DCE segment created for that user in RACF
   - The AUTOLOGIN variable in the DCE segment set to NO

   This is necessary whether the server is started by batch job or by a procedure. Configuring this user ID differently could produce unpredictable results when the server is started.

Overview of RACF Interoperability

In order to have interoperability, the RACF database must contain information that associates an OS/390-RACF user ID with a DCE principal and the DCE principal's UUID with the corresponding OS/390-RACF user ID. The interoperability information is contained in a new RACF DCE segment and in the RACF general resource class, DCEUUIDS. The RACF profile for a given OS/390 user ID has been enhanced to contain an optional DCE segment. After this RACF DCE segment is created for an OS/390 user ID by a RACF administrator and fully populated, it contains DCE information for that user, including user principal name, principal UUID, DCE cell name, DCE cell UUID, and automatic login flag. The segment also contains the DCE principal's password, which is used for single sign-on. (More about single sign-on in “Single Sign-on for OS/390 and DCE” on page 409). The information in the DCE segment is used both for single sign-on and for determining a DCE identity when a corresponding RACF identity is known.

The information placed in both the RACF DCE segment and the RACF general resource class, DCEUUIDS, is called **cross-linking information**. This cross-linking information must be set up before interoperability functions can be used.

RACF provides administrator commands for creating, modifying, and deleting the new RACF user profile DCE segment and RACF general resource class, DCEUUIDS. See the *OS/390 Security Server (RACF) Security Administrator's Guide* for information on these commands.

OS/390 DCE provides the utilities that support RACF interoperability. These utilities are discussed next.

The RACF Interoperability Utilities

Two OS/390 DCE utilities, **mvsimpt** and **mvsexpt**, are provided to automate much of the administrator's work in creating the cross-linking information. The **mvsexpt** utility creates the cross-linking information in the RACF database from information in the DCE registry. It also allows the migration of the OS/390 DCE Application Support sidefile. The **mvsimpt** utility creates DCE principals from information obtained from the RACF database.

The **mvsexpt** utility populates individual RACF users' DCE segments with the corresponding DCE principal name, principal UUID, cell name, cell UUID, and AUTOLOGIN setting. It also populates the RACF general resource class, DCEUUIDS, with an entry mapping the individual DCE principal's UUID and OS/390 user ID.

While the RACF DCE segment contains the DCE principal's password, the utilities do not fill in the password field of the DCE entry. This must be done by each individual user after the DCE segment is created and populated, and any time the user changes his or her DCE password. The use of the DCE segment information is primarily for single sign-on, but can also be used whenever information in a user's DCE segment is needed. For more on this topic, see “Single Sign-on for OS/390 and DCE” on page 409.

The utilities handle these categories of users:

- The user is an existing RACF user, but is a new DCE user (see “Cross Linking Existing RACF Users who are New DCE Users” on page 400)
- The user is an existing DCE user, but is a new RACF user (see “Cross Linking Existing DCE Users who are New RACF Users” on page 405)
- The user exists in both DCE and RACF, but the information needs to be cross linked (see “Cross Linking Existing DCE Users who are Existing RACF Users” on page 407).

The utilities are supported only on an OS/390 system and must be run on the OS/390 host system whose RACF database is to be cross linked. If the RACF users are principals in different cells, then the utilities must also be run against the DCE registry for each unique cell. For more information, see “Multi-Cell Considerations” on page 398. For security reasons, these utilities can only be run by RACF and DCE administrators. The utilities do not support RACF Remote Sharing.

The initial cross-linking can be done from either the RACF database or the DCE registry, but **mvsimpt** and **mvsexpt** must be run from the OS/390 system where the RACF database resides. Before the utilities can be run, the data entry for cross linking must be done by the RACF administrator (creating RACF DCE segments for OS/390 users) or the DCE cell administrator (creating DCE principals). As shown in Figure 63 on page 391, where the data entry is done (RACF or DCE) determines where the cross linking is initiated.

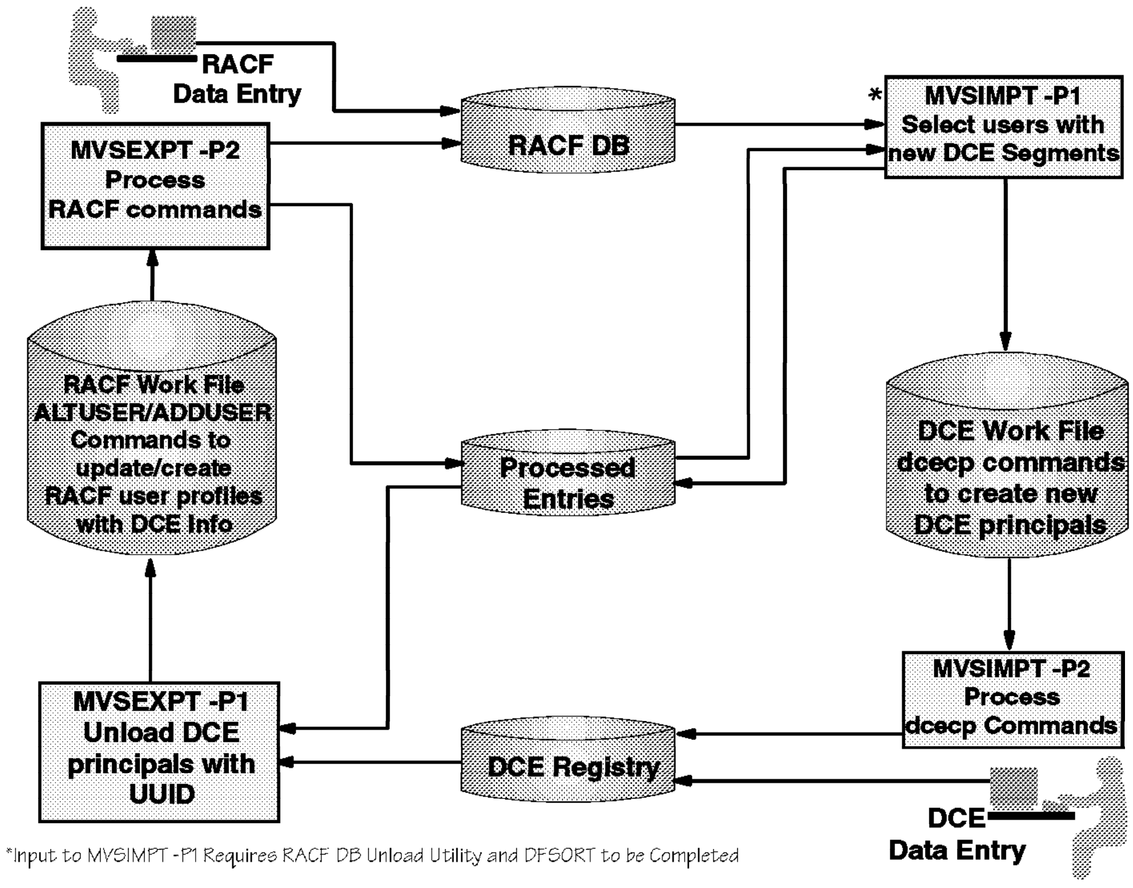

RACF Data Entry
MVSEXPT -P2 Process RACF commands
RACF DB
* MVSIMPT -P1 Select users with new DCE Segments
RACF Work File ALTUSER/ADDUSER Commands to update/create RACF user profiles with DCE Info
Processed Entries
DCE Work File dcecp commands to create new DCE principals
MVSIMPT -P2 Process dcecp Commands
MVSEXPT -P1 Unload DCE principals with UUID
DCE Registry
DCE Data Entry
*Input to MVSIMPT -P1 Requires RACF DB Unload Utility and DFSORT to be Completed

Figure 63. Cross-Linking Utilities. The input to the utilities comes from either the RACF database or the DCE registry.

The **mvsimpt** and **mvsexpt** utilities require that the users to be cross linked be first defined to either DCE or RACF. The process of defining or updating users to the DCE registry that have been previously defined to RACF is the **user importing process**. The process of defining or updating users to RACF that have been previously defined to the DCE registry is the **user exporting process**. (The concepts of import and export are seen from the perspective of the DCE registry.)

The import-export process can be seen as a circle (as in Figure 63), which can be entered from one of two points depending on whether existing DCE users are to be cross linked with OS/390 RACF IDs, or new DCE principals are to be cross linked with existing OS/390 RACF IDs.

Existing DCE Principals: If DCE principals exist, and your installation wishes to define a subset of them as OS/390 users, the DCE cell administrator, working with the RACF administrator, determines which DCE principals map to a subset of the OS/390-RACF users.

The DCE administrator then uses the DCE export utility, **mvsexpt**, which runs in two passes. When you run the first pass, it generates a set of RACF commands that will cross link the DCE principals to RACF user IDs in the RACF database on the second pass. This is the step labeled **mvsexpt -p1** in Figure 63.

How users are defined for export is determined by the input option provided when **mvsexpt** is run. One option , **-u**, provides **mvsexpt** with a Hierarchical File System (HFS) file name that contains the DCE principals to be cross linked to RACF. For other options see Figure 64 on page 401 and the *OS/390 DCE: Command Reference*.

The RACF administrator next reviews the output of **mvsexpt** pass one (to ensure that the commands created by the utility are what the administrator desires) and then runs the second pass. In the second pass, **mvsexpt** runs the RACF commands generated earlier. These commands modify existing RACF User Profiles or create new ones. The commands populate the user DCE segment of the RACF users. They also create the entry for each DCE principal's UUID in the RACF general resource class, DCEUUIDS, and the DCE principal's associated OS/390 user ID. The DCEUUIDS entries are done automatically as part of the RACF **adduser** and **altuser** commands.

Note: **mvsexpt -p2** creates a RACF user entry, including an OMVS segment, if one does not already exist. (An OMVS segment is a specific segment in RACF for OS/390. Its complete name is the User OMVS Data Record.) Default values are used, but the RACF administrator can change these values in the EXPTVAR file before pass one is run, if desired. See "Tailoring the Utilities for Your Environment" on page 393 for more information. The user's OMVS segment is required for access to OS/390 DCE services. (See the *OS/390 Security Server (RACF) Security Administrator's Guide*.)

The **mvsexpt** utility creates a log file, **/opt/dcelocal/var/security/adm/RACFERS**, that captures command failures associated with the RACF command stream. This file must be checked by both the DCE administrator and the RACF administrator to ensure the completeness of the principal-exporting process.

New DCE Principals: If your installation is creating new DCE principals, the DCE registry must be populated with DCE account information. If the new DCE principals are to be cross linked with existing OS/390 RACF IDs, the DCE registry can be populated using information from the RACF database. This is the entry point labeled **RACF Data Entry** in Figure 63 on page 391.

The RACF and DCE administrators identify the subset of users to be defined as DCE principals, and the RACF administrator creates a minimal DCE segment for each RACF user to be cross-linked with information from the DCE registry. To create a minimal DCE segment, type **altuser** *mvsid* **dce**.

The RACF administrator then creates the input file for the DCE import utility, **mvsimpt**. You can use a combination of the RACF Database unload utility, **IRRDBU00** and the IBM DFSORT Program Product (or equivalent), to create a data set and then copy it to an HFS file that contains the OS/390-RACF users that have DCE segments.

Note: Any sort utility can be used in this step, but the output must be standard. See Appendix G, "Files Created and Used by mvsimpt and mvsexpt" on page 541 for the format of the sorted output.

The DCE cell administrator runs the first pass of the **mvsimpt** utility, which builds the required DCE registry command streams to populate the DCE registry with the principals being created from information in the RACF database. After reviewing the output of pass one and making any changes needed, the DCE cell administrator next runs **mvsimpt** pass two to populate the DCE registry and reviews the output.

Because the DCE segment created by the RACF administrator is a minimal one, it must then be populated. The **mvsexpt** utility must be run to export data from the DCE registry into the RACF database for each principal to be cross linked. The **-e** option should be used when invoking the first pass of **mvsexpt** (after running **mvsimpt**). This option processes the entries previously processed by **mvsimpt**. Both passes of **mvsexpt** must be run. When pass two is completed, each DCE principal is cross linked with its corresponding RACF user ID in the RACF database. This is described in "Existing DCE Principals" on page 391.

Step-by-step instructions for the DCE-RACF cross-linking processes are discussed in “Cross Linking Existing RACF Users who are New DCE Users” on page 400, “Cross Linking Existing DCE Users who are New RACF Users” on page 405, and “Cross Linking Existing DCE Users who are Existing RACF Users” on page 407.

Tailoring the Utilities for Your Environment

DCE provides you, the administrator, the ability to tailor the utilities to your environment and to thereby obtain the desired results in the cross-linking process. Tailoring is accomplished by the editing of two files, **/opt/dcelocal/etc/IMPTVAR** for **mvsimpt** and **/opt/dcelocal/etc/EXPTVAR** for **mvsexpt**.

The **IMPTVAR** and **EXPTVAR** files each contain a set of variables which allow the administrator to:

- Control the processing of specific sets of users to ensure that the correct set of users are processed
- Control the parameters of the commands that are generated to populate the RACF database and DCE registry.

Note: All variable fields are required to have a value. Any missing values will cause the program to halt execution with an error message.

Tailoring Variables Common to mvsimpt and mvsexpt

The following variables are used by both **mvsimpt** and **mvsexpt** for tailoring:

- **CELLNAM**

  This variable is used by both utilities to indicate which cell should be processed for a given set of users. Because each RACF user can be enrolled in a different cell, this variable ensures that the correct set of users are processed. This variable also ensures that the cell the administrator is currently logged into matches the set of users to be processed. **CELLNAM** should be set to the name of the DCE cell where the set of users being processed exist or will exist. This variable should use a global cell name, for example, **/.../dcecell25.endicott.ibm.com**.

- **DEFCELL**

  This variable gives the cell name of the host principal (which is the principal for the machine where the RACF database resides). The utilities use this variable to ensure the proper processing of those users who do *not* have their **HOMECELL** explicitly defined in the RACF DCE segment. If the **HOMECELL** value is left blank in the RACF DCE segment then the DCE user's implicit **HOMECELL** is DEFCELL. **DEFCELL** should be set to the cell name of the host principal. This variable should use a global cell name, for example, **/.../dcecell25.endicott.ibm.com**.

Tailoring Variables for mvsimpt

The following variables are used for tailoring:

- **DCEGRP**

  DCEGRP sets the default DCE group for the DCE user principal. It sets the **-group** option on the **dcecp user create** command. An example is **-group Project**. To see how this is used, refer to Figure 99 on page 547.

- **DCEORG**

  DCEORG sets the default DCE organization for the DCE user principal. It sets the **-organization** option on the **dcecp user create** command. An example is **-organization None**. To see how this is used, refer to Figure 99 on page 547.

Tailoring Variables for mvsexpt

The following variables are used for tailoring:

- Variables for the RACF OMVS segment:
  - **UIDVAR**

    This variable sets the RACF OS/390 (OMVS) segment UID associated with the user. If a user is being added to RACF, an OMVS data record is created to allow the user to access OS/390 services. **mvsexpt -p1** creates the RACF commands necessary to accomplish this. See the *OS/390 Security Server (RACF) Security Administrator's Guide* for further information.

    This variable is set to an initial value and then incremented by one (for each successive user) for each invocation of the utility. Administrators should set this value, as required, for subsequent invocations, if duplicate UIDs are not desired. The setting of the user's OMVS UID is extremely important. See the *OS/390 UNIX System Services Planning* for further information.
  - **PROVAR**

    This variable sets the RACF OS/390 (OMVS) segment Default Program associated with the UID. If a user is being added to RACF, an OMVS data record is created to allow the user to access OS/390 services. **mvsexpt -p1** creates the RACF commands necessary to accomplish this. See the *OS/390 Security Server (RACF) Security Administrator's Guide* for further information.
- Variables for the RACF DCE segment:
  - **HOMECELL**

    This variable determines if the home cell name field, **HOMECELL**, in the RACF DCE segment is required. **HOMECELL** can be set to either **YES** or **NO**. If **YES** is specified, the **HOMECELL** value in the RACF DCE segment contains the cell name being processed. If **NO** is specified, the **HOMECELL** value in the RACF DCE segment contains no cell name.
  - **HOMEUUID**

    This variable determines if the home cell UUID field, **HOMEUUID**, in the RACF DCE segment is required. **HOMEUUID** can be set to either **YES** or **NO**. If **YES** is specified, the **HOMEUUID** value in the RACF DCE segment contains the home cell UUID being processed. If **NO** is specified, the **HOMEUUID** value in the RACF DCE segment contains no home cell UUID.

    Note: The **HOMEUUID** variable should be set to **YES** to ensure that a RACF-DCE user can be uniquely identified by the combination of cell UUID and principal UUID. This will ensure that applications can consistently and correctly retrieve the DCE principal's unique identity.
  - **AUTOLOGIN**

    This variable determines if the automatic login field, **AUTOLOGIN**, in the RACF DCE segment is required. **AUTOLOGIN** can be set to either **YES** or **NO**. If **YES** is specified, the **AUTOLOGIN** value in the RACF DCE segment contains the **AUTOLOGIN** flag set to YES. If **NO** is specified, the **AUTOLOGIN** value in the RACF DCE segment contains the flag set to NO.

A table is provided below which summarizes the settings for,

- **HOMECELL**
- **HOMEUUID**
- **AUTOLOGIN**

and their resulting RACF **adduser** and **altuser**, command parameters.

Table 27. Summary of variable file settings and their resulting RACF command parameters

Variable Name	Var. Set.	RACF adduser Command Keywords and Variables	RACF altuser Command Keywords and Variables
HOMECELL	YES	HOMECELL (*registry_cell*)	HOMECELL (*registry_cell*)
	NO	default is NO	NOHOMECELL
HOMEUUID	YES	HOMEUUID (*registry_cell_uuid*)	HOMEUUID (*registry_cell_uuid*)
	NO	default is NO	NOHOMEUUID
AUTOLOGIN	YES	AUTOLOGIN(YES)	AUTOLOGIN(YES)
	NO	default is NO	NOAUTOLOGIN

Note: The RACF adduser and altuser command parameters shown above are part of the RACF command to create a DCE segment. An example of this command is **altuser** *mvsid* **dce(homecell(***registry_cell***))**.

Tailoring the Location of Files for mvsimpt and mvsexpt: The variables in **/opt/dcelocal/etc/IMPTVAR** and **/opt/dcelocal/etc/EXPTVAR**, used to specify the path for working files, generally do not require tailoring unless the administrator desires different file paths.

Guidelines for Using mvsimpt and mvsexpt

The following guidelines are provided so that you may obtain the best results when using **mvsimpt** and **mvsexpt**:

- Only one DCE principal can be cross linked with a given RACF user in a RACF database.
- Each invocation of **mvsexpt** and **mvsimpt** causes the output and error files, except the Processed Entries file, to be nulled. Therefore, if any error files are to be corrected and the utilities rerun, be sure to run pass two of the utilities before rerunning pass one.
- DCE passwords must be changed after **mvsimpt** pass one is run if the resulting passwords do not conform to your organization's password rules.
- The **mvsexpt** utility does not provide the DCE password for an OS/390 user's DCE segment. The password is needed when an OS/390 user's RACF DCE segment AUTOLOGIN flag is set to **YES**. You must tell each individual user to set his or her password after the DCE segment is created and populated, and any time the user changes the DCE password. Setting or changing the password is done using the **storepw** command. For more information, see “Single Sign-on for OS/390 and DCE” on page 409.
- Follow in proper order the steps for invoking each utility, including any setup or setting of variables that must be done in the **/opt/dcelocal/etc/IMPTVAR** and **/opt/dcelocal/etc/EXPTVAR** files.

  Note: How certain variables are set in these files is important because they cannot be changed later. See the “do not change” list items below.
- Run the passes in correct sequence.

  mvsimpt and **mvsexpt** are command generator and processing utilities. Each utility has a pass one that generates the commands for pass two to process. Each utility must be run in sequence, that is, pass one must be run first followed by pass two. Also, if **mvsimpt** was run, **mvsexpt** with the **-e** option must be run before **mvsexpt** can be run with any other option.

- After you have used **mvsimpt** and **mvsexpt** the first time, from then on you must always use the utilities whenever you want to cross link DCE principals and OS/390 RACF users. If any other method is used, the Processed Entries file, **/opt/dcelocal/var/security/adm/PROCENTR**, which acts as a filter to remove already processed entries, will not be accurate, causing unpredictable results upon subsequent invocations of the utilities.
- Never update the Processed Entries File, **/opt/dcelocal/var/security/adm/PROCENTR**, except if instructed by utility documentation.
- For the **mvsimpt** utility,
  - Be sure the **mvsimpt** input file to pass one contains only the entries to be processed. Other entries should be deleted before invocation of pass one. Entries *must not* be deleted from the Pass One Output file **/opt/dcelocal/var/security/adm/DCEWORK**.
  - Be sure the principal name is specified in the RACF DCE segment if the default created by pass one is not the principal name wanted. The default is the OS/390 user ID.
  - If the RACF unload file, **/opt/dcelocal/var/security/adm/RACFUNLD** contains a UUID, **mvsimpt** assumes the user has already been cross linked and the entry will not be processed.
  - Do *not* change the values for any DCE principal name in the **/opt/dcelocal/var/security/adm/DCEWORK** file created by pass one. You can change any other value for data elements (such as group or org) on the commands in the **/opt/dcelocal/var/security/adm/DCEWORK** file created by pass one.
- For the **mvsexpt** utility,
  - Entries must not be deleted from the Pass One Output file, **/opt/dcelocal/var/security/adm/RACFWORK**.
    - Do *not* change any of the following data elements in the **/opt/dcelocal/var/security/adm/RACFWORK** file created by pass one:
      - Any OS/390 user ID
      - Any DCE principal name
      - Any UUID
      - The **HOMECELL** value

      You can change values for any *other* data elements, such as OMVS segment information. (Remember, how you set these four variables in the **/opt/dcelocal/etc/EXPTVAR** file was an important step, because they must not be changed later. They must not be changed in the Processed Entries file either.)

Failure to comply with any of the above guidelines can cause unpredictable utility results, such as incorrect RACF DCE segment information.

Considerations for the mvsimpt Utility: The **mvsimpt** utility creates a DCE principal in the DCE registry, if one does not already exist.

The **mvsimpt** utility is a command that must be entered from the OS/390 UNIX shell. When **mvsimpt** is run, **mvsexpt** must be run to complete the cross-linking process. The person running the **mvsimpt** utility must be a DCE cell administrator.

Before the DCE cell administrator runs the **mvsimpt** utility, the RACF administrator must:

- Create a minimal DCE segment for each RACF user to be cross linked to the DCE registry. For example, use the following RACF command to create a minimal segment:

  ```
  altuser os390id dce
  ```

  See "Cross Linking Existing RACF Users who are New DCE Users" on page 400 for a discussion on creating DCE RACF segments.
- Run the RACF database unload utility, **IRRDBU00**. (For more information on the RACF **IRRDBU00** utility, see the *OS/390 Security Server (RACF) Security Administrator's Guide*.)
- Run DFSORT (or an equivalent sorting program) using as input the unload data set created by the RACF database utility, **IRRDBU00**, sorting on users that have a DCE segment (record type 0290). For more information, see *DFSORT Application Programming Guide*, SC33-4035.
- Copy the output from the DFSORT data set into an HFS file named **/opt/dcelocal/var/security/adm/RACFUNLD**. This is the file **mvsimpt** uses as its input file.
- Delete any entries from the input file that you do not want to be processed by **mvsimpt**.

Prior to running the **mvsimpt** utility, the DCE administrator must:

- Log in to the DCE cell in which principals are to be created.
- Look at the **mvsimpt** variables input file, **/opt/dcelocal/etc/IMPTVAR**, changing any defaults necessary. **DEFCELL** and **CELLNAM** must be set. For information on **DEFCELL** and **CELLNAM**, see "Tailoring the Utilities for Your Environment" on page 393.

Considerations for the mvsexpt Utility: The **mvsexpt** utility creates or updates the RACF DCE segment for each DCE principal to be cross linked with the RACF database.

The **mvsexpt** utility is a two-pass command that must be entered from the OS/390 UNIX shell. The person running **mvsexpt** pass one must be a DCE Cell administrator. For **mvsexpt** pass two, it must be run by the RACF administrator.

Prior to running the **mvsexpt** utility, the DCE administrator must:

- Log in to the DCE cell from which the DCE principals are to be cross linked.
- Enroll all DCE principals to be cross linked with RACF in the DCE registry. This can be done using the OS/390 DCE commands or the **mvsimpt** utility.
- Make any changes to the **mvsexpt** input variables file, **/opt/dcelocal/etc/EXPTVAR**. The variables **CELLNAM** and **DEFCELL** are used by **mvsexpt** to determine if the correct cell is being cross linked with RACF.

  The variables **HOMECELL**, **HOMEUUID**, and **AUTOLOGIN** determine which elements to include in the RACF DCE segment. See "Tailoring the Utilities for Your Environment" on page 393 for a discussion of these variables.
- Provide input to pass one from one of the following:
  - The DCE registry (the default), which uses all principals.
  - A DCE registry subset

    Note: A subset of the DCE registry can be created and specified using the **-u** option. To create a DCE registry subset, the DCE cell administrator can use the following steps:

    1. Enter the command:

       ```
       dcecp -c principal catalog -simplename > /opt/dcelocal/var/security/adm/PRNIDMAP
       ```

       This puts a list of all principals in the registry into the PRNIDMAP file.

2. Edit the PRNIDMAP file:

   - Delete those users not required
   - Add a blank line between each entry
   - If desired, add a specific OS/390 ID to be associated with a specific DCE principal ID.

   See Figure 108 on page 552 for the format the PRNIDMAP file must have for correct processing.

– Output from **mvsimpt** contained in the Processed Entries file, **/opt/dcelocal/var/security/adm/PROCENTR**, specified on the call to **mvsexpt** pass one with the **-e** option.

– A migration file in the form created as part of the Application Support optional feature of OS/390 DCE, specified on the call to **mvsexpt** pass one **-m** option.

  - In OS/390 DCE Application Support, the Application Support administrator was required to create an identity mapping input file containing the DCE principal name and the OS/390 user ID and optionally an Application Support server name. This input file was then supplied as input to a batch job that created an identity mapping output file. This side file created additional work for the Application Support administrator and required that any updates to either the DCE Security registry or the OS/390 security database (such as RACF), be reflected in the Application Support identity file.

    In the current release of Application Support, this side file mapping can optionally be replaced by the RACF-DCE cross-linking information in the RACF database. To make the migration from the OS/390 DCE Application Support side file to the RACF-DCE cross-linking information in RACF easier, the **mvsexpt** utility can take the OS/390 DCE Application Support side file as input.

    Note: This input option can also be used by any other product that uses an identity mapping file as defined by Application Support, where the object is to migrate to the RACF-DCE cross-linking information in the RACF database.

– A user principal mapping file, created by the DCE administrator, and specified on the call to **mvsexpt** pass one (**-u** option). This file is similar to the Application Support migration side file. It may contain an associated OS/390 user ID to be created if the default OS/390 ID is not wanted. The default OS/390 ID is the first one to seven valid OS/390 characters of the DCE principal ID. If no valid OS/390 ID can be generated this way, the utility creates an ID that consists of the prefix “DCE” and a four-digit random number.

  Note: The principal mapping file, **/opt/dcelocal/var/security/adm/PRNIDMAP**, (or similar file) can be created by the administrator manually or by following the steps used for a DCE registry subset in the note on how to create a registry subset on page 397.

Multi-Cell Considerations

Each RACF user can be enrolled in a different cell. The RACF and DCE administrators should consider the following for these cases:

- Any time you are processing multiple cells
  – Because many of the input and output files are shared between **mvsimpt** and **mvsexpt**, and the output files are nulled at the invocation of each utility, one cell should be fully processed before processing is begun on the next cell.
- When starting cross linking with the RACF database:
  – Provide the cell name when creating RACF DCE segments

  The RACF administrator can create the DCE segments for all RACF users to be cross linked, even when multiple cells are involved. The cell name must be provided when a DCE segment is created for those users who belong to a cell different from the default cell. For a discussion of default cell, see "Tailoring the Utilities for Your Environment" on page 393.

  - Create the input file for the **mvsimpt** utility

    Run the RACF database unload utility IRRDBU00 and the sort program DFSORT (or equivalent) sorting on DCE segments (database record 290). Using the sorted file as input, **mvsimpt** must be tailored and run against each DCE cell registry for which a RACF user ID listed in the input file has a DCE principal.

  - Tailor the utilities

    The **/opt/dcelocal/etc/IMPTVAR** file contains variables that determine which cell is being processed and which users from the input file are to be processed. These variables must be changed for each DCE cell that is to be processed with **mvsimpt**. For a discussion of each of these variables, see "Tailoring the Utilities for Your Environment" on page 393.

  - Log into the DCE cell

    Before invoking **mvsimpt**, the DCE administrator must log in to the DCE cell from the OS/390 DCE system where the RACF users are defined who are to be enrolled as DCE principals.

  - Tailor the utilities

    The **/opt/dcelocal/etc/EXPTVAR** file contains variables that determine which cell is to be processed and which users from the input file are to be processed. These variables must be changed to match those set in **/opt/dcelocal/etc/IMPTVAR**.

    Note: The **HOMEUUID** variable should be set to **YES** to ensure that a RACF-DCE user can be uniquely identified by the combination of cell UUID and principal UUID. This will ensure that applications can consistently and correctly retrieve the DCE principal's unique identity.

  - Run **mvsexpt -p1 -e**

    This creates a RACF command file with the proper parameters for processing by pass two.

  - Run **mvsexpt -p2**

    This processes the RACF database using the RACF command file and completes the cross linking process.

  - Repeat the process for individual cells.

    For each cell that has one or more DCE principals to be cross linked to the RACF database, tailor and run **mvsimpt**. Be sure to finish processing one cell (that is, run **mvsexpt**) before invoking **mvsimpt** again.

- When starting cross linking with the DCE registry:

  - Tailor the utilities

    The **/opt/dcelocal/etc/EXPTVAR** file contains variables that determine which cell is being processed and what values get placed in the RACF database. For a description of these values and what they mean, see "Tailoring the Utilities for Your Environment" on page 393.

  - Log into the DCE cell

    Before invoking **mvsexpt**, the DCE administrator must log in to the DCE cell from the OS/390 DCE system where the RACF users are cross linked with information from the DCE registry.

  - Run **mvsexpt** multiple times

For each cell that has one or more DCE principals to be cross linked to the RACF database, tailor and run **mvsexpt**. Be sure to run both passes of **mvsexpt** and handle all errors before invoking **mvsexpt** for another cell.

Cross Linking Small Numbers of Users

The utilities should be used even for enrolling a small number of users. Here are some tips for doing this:

- If the users do not exist as DCE principals, create the principals in the DCE registry using **dcecp** commands, such as **dcecp user create**.
- Create a principal file, optionally specifying the OS/390 user ID of the user to be created if the default is not wanted.
- Run **mvsexpt** pass one and two to cross link the user.

Specific steps for the whole process are given in “Cross Linking Existing RACF Users who are New DCE Users” and “Cross Linking Existing DCE Users who are New RACF Users” on page 405.

Recreating the Processed Entries File

If the Processed Entries file, **/opt/dcelocal/var/security/adm/PROCENTR** should be lost or corrupted, the RACF administrator can recreate the Processed Entries File by doing the following:

1. Run the RACF database unload utility, **IRRDBU00**.
2. Run DFSORT or equivalent, sorting on RACF users that have a DCE segment.
3. Edit the output to contain only those entries whose principal name, UUID, and (optionally) **HOMECELL** are filled in.
4. Delete the record number so that the OS/390 user ID begins in column 1. Maintain relative positioning of other fields.
5. Copy the edited data set to the Processed Entries File, **/opt/dcelocal/var/security/adm/PROCENTR**, creating the file if it does not exist.

Introduction to Administration Scenarios

This section takes you step by step through the cross-linking process, regardless of whether the process is started from the RACF database or the DCE registry.

The following scenarios are described:

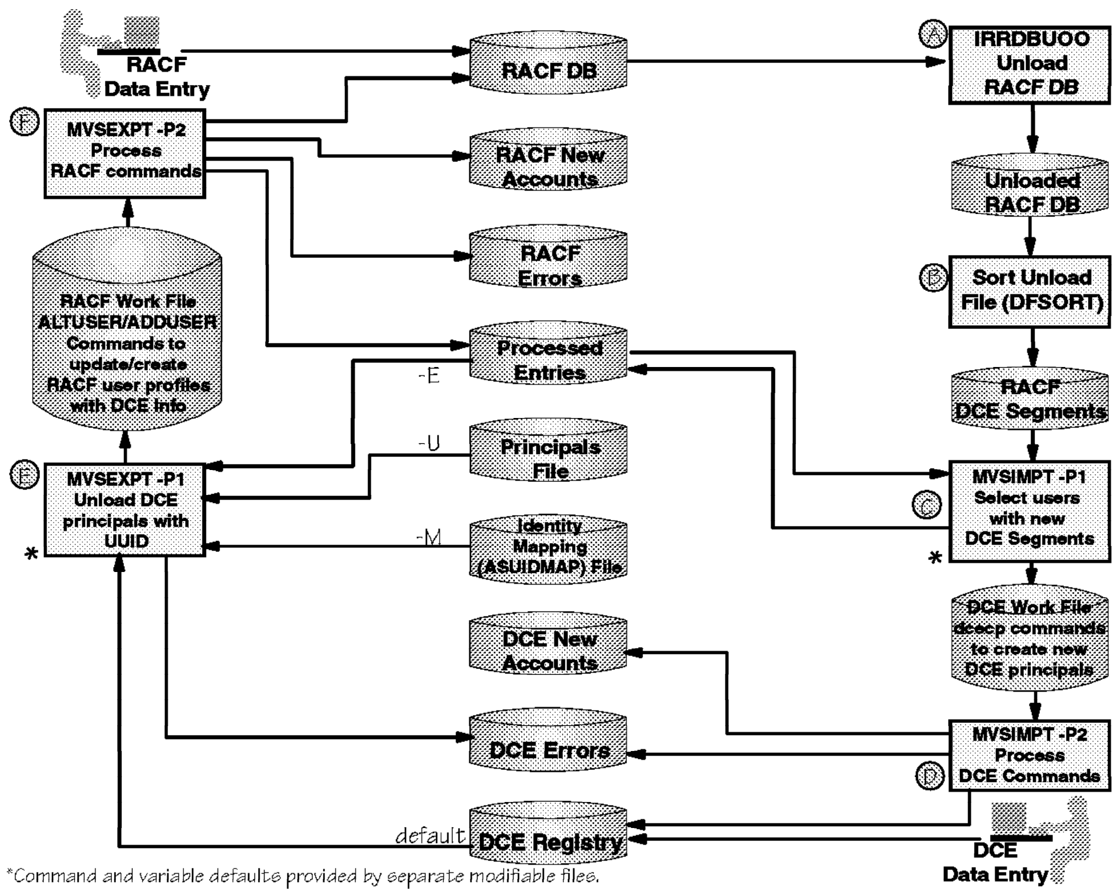
- Cross linking existing RACF users who are new DCE users
- Cross linking existing DCE users who are new RACF users
- Cross linking existing DCE users who are existing RACF users

Cross Linking Existing RACF Users who are New DCE Users

Do the following steps to populate the RACF database and DCE registry, starting with information in the RACF database:

1. Create DCE segments in RACF.

   The upper left corner of Figure 64 on page 401 shows data entry being done by a RACF administrator on the RACF database. The RACF administrator is creating a DCE segment for each RACF user that is to be cross linked with this DCE registry.



RACF Data Entry
RACF DB
A
IRRDBUOO Unload RACF DB
F
MVSEXPT -P2 Process RACF commands
RACF New Accounts
Unloaded RACF DB
RACF Errors
B
Sort Unload File (DFSORT)
RACF Work File ALTUSER/ADDUSER Commands to update/create RACF user profiles with DCE Info
Processed Entries
-E
RACF DCE Segments
-U
Principals File
E
MVSEXPT -P1 Unload DCE principals with UUID
*
-M
Identity Mapping (ASUIDMAP) File
MVSIMPT -P1 Select users with new DCE Segments
C
*
DCE Work File dcecp commands to create new DCE principals
DCE New Accounts
DCE Errors
MVSIMPT -P2 Process DCE Commands
D
default
DCE Registry
DCE Data Entry
*Command and variable defaults provided by separate modifiable files.

Figure 64. Detailed View of RACF-DCE Cross-Linking Utilities. Using cross-linking utilities from the RACF database or the DCE registry.

The RACF administrator defines the DCE segment using RACF commands. (For these commands and related information, see *OS/390 Security Server (RACF) Security Administrator's Guide*.) The RACF administrator should provide the minimum amount of information needed when creating a DCE segment. The DCE utilities **mvsimpt** and **mvsexpt** use the tailored information you provide (see “Tailoring the Utilities for Your Environment” on page 393 for more information), and then supply the cross-linking information needed. At a minimum, the RACF administrator only needs to create the DCE segment without supplying any DCE segment parameters. To create a minimal DCE segment, type **altuser** *os390id* **dce**. Whenever possible, this is the way the segment should be created. However, there are times when one or two parameters may be needed when creating the DCE segment:

- **DCENAME**, the DCE user principal name, if the default principal name created by the utility is not wanted. The default is the OS/390 user ID.
- **HOMECELL**, if *either* of these is true:
  - The DCE **HOMECELL** cell name is different from the default cell name. The default cell is defined to be the cell name of the host principal for the machine where the RACF database resides.
  - The DCE **HOMECELL** cell name is the same as the default cell name and the RACF administrator wants it in each user's RACF DCE segment. It is suggested that the cell name

only be provided if different from the default cell name to avoid unnecessary changes to the RACF DCE segment entry.

Also, an entry must be made for the mapping between the DCE principal's UUID and the RACF user ID in the RACF General Resource Class, DCEUUIDS. This entry is created for the RACF administrator when the RACF **adduser** and **altuser** commands are run.

Note: Only one DCE principal can be cross linked with a given RACF user ID. If a second principal is specified by means of a second RACF ALTUSER command, the current principal name gets overwritten. The RACF administrator must make sure that the other fields in the DCE segment are changed appropriately. If the cell name of the DCE segment is changed for users that have already been processed by these utilities, the RACF administrator must update the **HOMECELL** field of the Processed Entries file if the **HOMECELL** field contains the previous cell name.

2. Run the RACF database unload utility, **IRRDBU00**.

   This utility unloads the RACF database into a data set which can then be used as input to a sort program. This is shown in Figure 64 on page 401, step A. For more information on how to run this utility and on the output file it creates, see the *OS/390 Security Server (RACF) Security Administrator's Guide*.

3. Run the **DFSORT** program.

   In Figure 64 on page 401 step B, the RACF administrator next runs **DFSORT** using the data set created by **IRRDBU00** as input to the utility. Sort the file by RACF database record 290. For more information about sorting criteria and the required format of the output file, see the *DFSORT Application Programming Guide*, SC33-4035.

   Note: Other sorting utilities can be used instead of **DFSORT**. However, the resulting sorted file must look like that created by using **DFSORT**, see Appendix G, “Files Created and Used by mvsimpt and mvsexpt” on page 541.

4. Copy the output from **DFSORT** into an HFS file named **/opt/dcelocal/var/security/adm/RACFUNLD**. This is the file **mvsimpt** uses as its input file.

5. From the OS/390 system where the RACF database containing the DCE segments resides, log on as DCE cell administrator to the DCE cell for which DCE principals are to be enrolled and cross linked.

6. Look at the **mvsimpt** variable input file, **/opt/dcelocal/etc/IMPTVAR**, and change any of the default values, if necessary. See “Tailoring the Utilities for Your Environment” on page 393 for a discussion of the values to set.

7. Run the **mvsimpt** utility specifying pass one and your DCE cell administrator password. If your password is *mypwd*, you would type:

   ```
   mvsimpt -p1 -pw mypwd
   ```

   from an OS/390 UNIX shell in an interactive session. See Figure 64 on page 401, step C. Input to this command is the HFS file, **/opt/dcelocal/var/security/adm/RACFUNLD**, which contains the output from **DFSORT**.

   Note: If the DCE-RACF utilities have been previously run, then **mvsimpt** uses the Processed Entries file to remove any entries from the input file that were previously processed.

   mvsimpt produces an output file, **/opt/dcelocal/var/security/adm/DCEWORK**, that contains DCE **dcecp** command arguments for creating corresponding principal entries in the DCE registry that the DCE administrator is currently logged in to. This file also contains the administrator's password so be sure that the file has proper protection.

8. Look at the output from pass one.

   mvsimpt pass one populates:

- The Pass One Output file

  The output of the first pass of **mvsimpt** goes into the Pass One Output file, **/opt/dcelocal/var/security/adm/DCEWORK**, an HFS file. For more details on this file, see Appendix G, “Files Created and Used by mvsimpt and mvsexpt” on page 541.

  After you run the first pass of **mvsimpt**, this file contains entries for only those RACF users that are to be cross populated with the DCE registry.

  Note: The entries processed during the first pass of **mvsimpt** are those whose cell name (or default cell name in **/opt/dcelocal/etc/IMPTVAR**) matches the cell name that the DCE administrator is currently logged in to. If there are other entries that belong to a different cell, **mvsimpt** must be run again (after running **mvsimpt** pass two and **mvsexpt**), with the DCE administrator logged in to the cell for which the entries are to be processed.

  Edit this file and make any changes to information such as group, organization, or user password, as needed. Individual DCE principal names and RACF user IDs must not be changed. For information on what can be changed, see “Guidelines for Using mvsimpt and mvsexpt” on page 395.

  The principal and an account are created by **mvsimpt** pass two. The principal is added to the group and organization specified in this file. For more information about the defaults taken, see “Tailoring the Utilities for Your Environment” on page 393. Entries must not be deleted. Input to pass one should contain all entries to be processed.

- Processed Entries file

  This HFS file, **opt/dcelocal/var/security/adm/PROCENTR**, created by **mvsimpt**, contains a list of the RACF user IDs and DCE principals to be cross linked in the RACF database. For each entry, the RACF user ID and DCE principal fields are filled in. The cell name is filled in if it was filled in the RACF DCE segment. The DCE principal's UUID is filled in during **mvsexpt** pass two. This file must not be edited or deleted.

  If, in the future, new DCE segments are added to RACF, and they need to be cross linked to the DCE registry, this file is used to filter out any users previously processed by this utility from the list of users to be processed by the next invocation of **mvsimpt** pass two. If this file is deleted and the **mvsimpt** or **mvsexpt** utility is run, entries already processed are reprocessed by the utility with varying results.

9. Run **mvsimpt** specifying pass two. Type:

   ```
   mvsimpt -p2
   ```

   from an OS/390 UNIX shell in an interactive session. This is step D in Figure 64 on page 401.

10. Look at the output from the utility.

    mvsimpt pass two populates two output files:

    - DCE Error file

      This HFS file, **/opt/dcelocal/var/security/adm/DCEERS**, contains all commands that **mvsimpt** pass two could not process along with an error indication. This file can be edited to fix the error and rerun. Some of the error entries may be because of the principal already existing in the DCE registry. For this case, the error can be ignored. If any records need to be reprocessed, fix the problem and delete the error indication. Delete any commands and error indications which you do not want to retry. Run pass two again specifying the **-r** option which uses the **/opt/dcelocal/var/security/adm/DCEERS** file as input.

    - The DCE New Accounts file

      This HFS file, **/opt/dcelocal/var/security/adm/DCENEW**, contains an entry for every DCE principal created and the principal's password. The DCE cell administrator should give each DCE user an initial password. Each new DCE principal is required to change this DCE password when first logging in to DCE. If the RACF user's DCE segment will be enabled for single sign-on, (AUTOLOGIN=YES set by **mvsexpt** pass two), remind the user to enter the OS/390 DCE **storepw** command to save his or her changed DCE password in the RACF DCE segment. This must also be done every time the DCE password is changed. The **storepw** command is described in the *OS/390 DCE: Command Reference*.

11. Look at the **mvsexpt** variable input file, **/opt/dcelocal/etc/EXPTVAR**, and change any default values, if necessary. See “Tailoring the Utilities for Your Environment” on page 393 for a discussion of values to set.

    Note: The **HOMEUUID** variable should be set to **YES** to ensure that a RACF-DCE user can be uniquely identified by the combination of cell UUID and principal UUID. This will ensure that applications can consistently and correctly retrieve the DCE principal's unique identity.

12. Run **mvsexpt** specifying pass one and the entries option. Type:

    ```
    mvsexpt -p1 -e
    ```

    In Figure 64 on page 401 step E, **mvsexpt** pass one option **-e**, takes the Processed Entries records created from running the **mvsimpt** utility, and creates the RACF commands to add the DCE principal's UUID into the RACF database. This output is in the HFS file, **/opt/dcelocal/var/security/adm/RACFWORK**, created by **mvsexpt**.

    A DCE Error file, **/opt/dcelocal/var/security/adm/DCEERS**, may be created as a result of executing **mvsexpt -p1**. This file contains failed requests to obtain a principal's UUID from the DCE registry. These errors must be resolved by one of the following means:

    - Create the principals in the DCE registry and rerun **mvsexpt -p1**
    - Edit the processed entries and RACF work files, **/opt/dcelocal/var/security/adm/PROCENTR** and **/opt/dcelocal/var/security/adm/RACFWORK**, removing the failing entries.

13. Run **mvsexpt** pass two. This must be done by the RACF administrator. Type:

    ```
    mvsexpt -p2
    ```

    In Figure 64 on page 401 step F, **mvsexpt** pass two updates information in the RACF DCE segments for the DCE principals processed by **mvsimpt**. For each DCE principal processed, it updates the DCE principal name, principal UUID, cell name, cell UUID, and AUTOLOGIN fields in the RACF DCE segment. It also creates an entry for this DCE principal in the RACF general resource class, DCEUUIDS, and populates it with the DCE principal's UUID and corresponding OS/390 user ID.

14. Look at the output from **mvsexpt** pass two.

    mvsexpt creates or updates the following HFS output files (for more information, see Appendix G, “Files Created and Used by mvsimpt and mvsexpt” on page 541):

    - RACF Error file

      This HFS file, **/opt/dcelocal/var/security/adm/RACFERS**, is created by **mvsexpt** and emptied at each new invocation of **mvsexpt**. Errors should not be encountered at this step, as DCE segments were already created prior to running **mvsimpt**. However, if any errors occur, the failing command along with the error indication is contained in this file. For any command that must be reprocessed, correct the error and delete the error indication. Delete any other messages or records that are not to be reprocessed. Run **mvsexpt** again, specifying the **-r** option, which uses **/opt/dcelocal/var/security/adm/RACFERS** as input.

    - Processed Entries file

      For each uncompleted entry in the Processed Entries file, **/opt/dcelocal/var/security/adm/PROCENTR**, that was processed successfully, the DCE principal's UUID field is updated. Entries that were not completed successfully are removed from the Processed Entries file and can be found in the RACF Error file. It is important that failed entries be resolved before running **mvsexpt -p2** again because the RACF error file is nulled upon subsequent invocations of **mvsexpt -p2**..

The Processed Entries file must not be edited or deleted. If, in the future, new DCE segments are added to RACF, and they need to be cross linked to the DCE registry, this file is used by **mvsimpt** pass one to filter out any users previously processed by this utility. If this file is deleted and the **mvsimpt** or **mvsexpt** utility is run, entries already processed are reprocessed by the utility with varying results.

The cross-linking process is now complete.

Cross Linking Existing DCE Users who are New RACF Users

These instructions assume that you are the DCE administrator, except as noted. Do the following steps to populate the RACF database, starting with information in the DCE registry:

1. Determine which input option to use with **mvsexpt** pass one.

   Based on the list of DCE principals to be cross linked, decide which of the following to use as input to **mvsexpt** pass one:

   - The entire DCE registry
   - A DCE registry subset

     Note: A subset of the DCE registry can be created and specified using the **-u** option. To create a DCE registry subset, the DCE cell administrator can use the following steps:

     a. Enter the command:

        ```
        dcecp -c principal catalog -simplename > /opt/dcelocal/var/security/adm/PRNIDMAP
        ```

        This puts a list of all principals in the registry into the PRNIDMAP file.

     b. Edit the PRNIDMAP file:

        – Delete those users not required

        – Add a blank line between each entry

        – If desired, add a specific OS/390 ID to be associated with a specific DCE principal ID.

     See Figure 108 on page 552 for the format the PRNIDMAP file must have for correct processing.

   - A principal mapping file

     Note: The principal mapping file, **/opt/dcelocal/var/security/adm/PRNIDMAP**, (or similar file) can be created by the administrator manually or by following the steps used for a DCE registry subset in the note on how to create a registry subset on page 397.

   For more information on these files, see “Considerations for the mvsexpt Utility” on page 397.

2. Look at the **mvsexpt** variable file, **/opt/dcelocal/etc/EXPTVAR**, and change any of the default values if necessary. See “Tailoring the Utilities for Your Environment” on page 393 for a discussion of these values.

   Note: The **HOMEUUID** variable should be set to **YES** to ensure that a RACF-DCE user can be uniquely identified by the combination of cell UUID and principal UUID. This will ensure that applications can consistently and correctly retrieve the DCE principal's unique identity.

3. Log in to the DCE cell

The DCE cell administrator must log into the DCE cell whose DCE principals are to be cross linked with the corresponding RACF users in the RACF database.

4. Run pass one of the DCE export utility, **mvsexpt**, specifying the correct input file option of your choice. In the following example, the DCE Cell Administrator wants all principals in the DCE registry to have a RACF user ID created or modified with a DCE segment. When the DCE registry is used as input, type:

   ```
   mvsexpt -p1
   ```

   Figure 64 on page 401 step E shows the **mvsexpt** process. The first pass of **mvsexpt** creates an HFS file, **/opt/dcelocal/var/security/adm/RACFWORK**, with RACF commands for all DCE principals listed in the input file. The command contains information to create a RACF DCE segment with the principal's name, UUID, and possibly cell name and cell UUID. (Values were set in **/opt/dcelocal/etc/EXPTVAR**.) If an OS/390 user ID is not provided, one is created. The default OS/390 ID is the first one to seven valid OS/390 characters of the DCE principal ID. If no valid OS/390 ID can be generated this way, the utility creates an ID that consists of the prefix “DCE” and a four-digit random number.

5. Look at the output from the utility.

   mvsexpt creates the RACF command input file. This HFS file, **/opt/dcelocal/var/security/adm/RACFWORK**, is populated by pass one of the **mvsexpt** utility. It is then input to pass two of the utility that runs the RACF commands. These commands update information in the RACF DCE segments and create entries in the RACF General Resource Class, DCEUUIDS, completing the cross-linking process. This file should not need editing and not all values can be changed. For a list of allowable changes, see “Guidelines for Using mvsimpt and mvsexpt” on page 395.

   A DCE Error file, **/opt/dcelocal/var/security/adm/DCEERS**, may be created as a result of executing **mvsexpt -p1**. This file contains failed requests to obtain a principal's UUID from the DCE registry. These errors must be resolved by one of the following means:

   - Create the principals in the DCE registry and rerun **mvsexpt -p1**
   - Modify the input file, **/opt/dcelocal/var/security/adm/PRNIDMAP** or a user specified file, to remove these entries and then rerun **mvsexpt -p1**
   - Edit the processed entries and RACF work files, **/opt/dcelocal/var/security/adm/PROCENTR** and **/opt/dcelocal/var/security/adm/RACFWORK**, removing the failing entries.

6. Run **mvsexpt** pass two. This must be done by the RACF administrator. Type:

   ```
   mvsexpt -p2
   ```

   In Figure 64 on page 401 step F, **mvsexpt** creates or updates a RACF DCE segment for each DCE principal in the **/opt/dcelocal/var/security/adm/RACFWORK**. For each DCE principal processed, it adds the DCE principal name, principal UUID, cell name, cell UUID, and AUTOLOGIN setting. If an OS/390 user ID is not provided, one is created. The default OS/390 ID is the first one to seven valid OS/390 characters of the DCE principal ID. If no valid OS/390 ID can be generated this way, the utility creates an ID that consists of the prefix “DCE” and a four-digit random number. The **mvsexpt** utility creates the RACF user ID entry, an OMVS segment, and DCEUUIDS general resource class.

7. Look at the output from **mvsexpt** pass two.

   mvsexpt pass two populates the following HFS output files created by **mvsexpt** (for more information on these files, see Appendix G, “Files Created and Used by mvsimpt and mvsexpt” on page 541):

   - RACF Error file

     Errors encountered running this step are placed in the HFS file, **/opt/dcelocal/var/security/adm/RACFERS**, created by **mvsexpt**. Each failing command along with the error indication is contained in this file. For any record that must be reprocessed, fix the error and delete the error indication. Delete any other messages or records that are not to be reprocessed. Run **mvsexpt** pass two again, specifying the **-r** option which uses **/opt/dcelocal/var/security/adm/RACFERS** as input.

- Processed Entries file

  The Processed Entries HFS file, **/opt/dcelocal/var/security/adm/PROCENTR**, contains entries that have been processed by the **mvsexpt** utility. This file must not be edited or deleted because it is read by subsequent invocations of the **mvsimpt** (pass one) or **mvsexpt** (pass one) utility to filter out users already processed. If this file is deleted and the **mvsimpt** or **mvsexpt** utility is run, entries already processed are reprocessed by the utility with varying results. Entries that were not completed successfully are removed from the Processed Entries file and can be found in the RACF Error file. It is important that failed entries be resolved before running **mvsexpt -p2** again because the RACF error file is nulled upon subsequent invocations of **mvsexpt -p2**..

- RACF New Accounts file

  This HFS file, **/opt/dcelocal/var/security/adm/RACFNEW**, is created by **mvsexpt**. An entry is made for each RACF user created. See Appendix G, "Files Created and Used by mvsimpt and mvsexpt" on page 541 for more information about the contents of this file. The RACF Administrator should contact each new user giving any information necessary. These RACF users may have to change their passwords the first time they log in to OS/390.

The cross linking process is now complete.

Cross Linking Existing DCE Users who are Existing RACF Users

These instructions assume that you are the DCE administrator, except as noted. Do the following steps to populate the RACF database, starting with information in the DCE registry:

1. Determine which input option to use with **mvsexpt** pass one.

   Based on the list of DCE principals to be cross linked, decide which of the following to use as input to **mvsexpt** pass one:

   - The entire DCE registry

     Using the registry generates RACF **adduser** commands that require changing to RACF **altuser** commands (see Step 5 on page 408). RACF OMVS segment commands are also generated that should be deleted if previously created. The RACF commands contain an OS/390 ID. The default OS/390 ID is the first one to seven valid OS/390 characters of the DCE principal ID. If the OS/390 ID generated is not desired, the administrator can do either of the following :

     a. Edit the **/opt/dcelocal/var/security/adm/RACFWORK** file and change the OS/390 IDs.

     b. Create a principal mapping file (see the note on using a subset of the DCE registry in "Considerations for the mvsexpt Utility" on page 397).

   - An identity mapping file

     Using the identity mapping file generates RACF **altuser** commands.

   - A principal mapping file

     Using the principal mapping file generates RACF **adduser** commands that require changing to RACF **altuser** commands (see Step 5 on page 408). RACF OMVS segment commands are also generated. These should be deleted if previously created. If the OS/390 ID is not supplied in the file, the RACF commands contain an OS/390 ID. The default OS/390 ID is the first one to seven valid OS/390 characters of the DCE principal ID. If the OS/390 ID generated is not desired, the administrator can edit the principal mapping file, **/opt/dcelocal/var/security/adm/PRNIDMAP**, and provide the specific OS/390 ID to be associated with a given principal user ID.

2. Look at the **mvsexpt** variable file, **/opt/dcelocal/etc/EXPTVAR**, and change any of the default values if necessary. See “Tailoring the Utilities for Your Environment” on page 393 for a discussion of these values.
3. Log in to the DCE cell

   The DCE cell administrator must log into the DCE cell whose DCE principals are to be cross linked with the corresponding RACF users in the RACF database.
4. Run pass one of the DCE export utility, **mvsexpt**, specifying the correct input file option of your choice. In the following example, the DCE Cell Administrator wants all principals in the DCE registry to have a RACF user ID modified with a DCE segment. When the DCE registry is used as input, type:

   ```
   mvsexpt -p1
   ```

   Figure 64 on page 401 step E shows the **mvsexpt** process. The first pass of **mvsexpt** creates an HFS file, **/opt/dcelocal/var/security/adm/RACFWORK**, with RACF commands for all DCE principals listed in the input file. The command contains information to create or update a RACF DCE segment with the principal's name, UUID, and possibly cell name and cell UUID. (Values were set in **/opt/dcelocal/etc/EXPTVAR**.)
5. Look at the output from the utility.

   mvsexpt creates the RACF command input file for pass two. This HFS file, **/opt/dcelocal/var/security/adm/RACFWORK**, is populated by pass one of the **mvsexpt** utility. It is then input to pass two of the utility that runs the RACF commands to update information in the RACF DCE segments and create entries in the RACF General Resource Class, DCEUUIDS, completing the cross-linking process. Depending on the input option specified on pass one, **mvsexpt** may have created RACF commands to create new users (that is, one or more RACF **adduser** commands). If this has occurred, change all **adduser** commands in the **/opt/dcelocal/var/security/adm/RACFWORK** file to **altuser** commands. Also, delete the OMVS segment part of each command if it has been created previously. The rest of this file should not need editing and not all values can be changed. For a list of allowable changes, see “Guidelines for Using mvsimpt and mvsexpt” on page 395.

   A DCE Error file, **/opt/dcelocal/var/security/adm/DCEERS**, may be created as a result of executing **mvsexpt -p1**. This file contains failed requests to obtain a principal's UUID from the DCE registry. These errors must be resolved by one of the following means:

   - Create the principals in the DCE registry and rerun **mvsexpt -p1**
   - Modify the input file, **/opt/dcelocal/var/security/adm/PRNIDMAP** or **/opt/dcelocal/var/security/adm/ASUIDMAP** or a user specified file, to remove these entries and then rerun **mvsexpt -p1**
   - Edit the processed entries and RACF work files, **/opt/dcelocal/var/security/adm/PROCENTR** and **/opt/dcelocal/var/security/adm/RACFWORK**, removing the failing entries.
6. Run **mvsexpt** pass two. This must be done by the RACF administrator. Type:

   ```
   mvsexpt -p2
   ```

   In Figure 64 on page 401 step F, **mvsexpt** updates a RACF DCE segment for each DCE principal in the **/opt/dcelocal/var/security/adm/RACFWORK**. For each DCE principal processed, it updates the DCE principal name, principal UUID, cell name, cell UUID, and AUTOLOGIN setting, if specified.
7. Look at the output from **mvsexpt** pass two.

   mvsexpt pass two populates the following HFS output files created by **mvsexpt** (for more information on these files, see Appendix G, “Files Created and Used by mvsimpt and mvsexpt” on page 541):

   - RACF Error file

     Errors encountered running this step are placed in the HFS file, **/opt/dcelocal/var/security/adm/RACFERS**, created by **mvsexpt**. Each failing command along with the error indication is contained in this file. For any record that should be reprocessed, fix the error and delete the error indication. Delete any other messages or records that are not to be reprocessed. Run **mvsexpt** pass two again, specifying the **-r** option which uses **/opt/dcelocal/var/security/adm/RACFERS** as input.

- Processed Entries file

  The Processed Entries HFS file, **/opt/dcelocal/var/security/adm/PROCENTR**, contains entries that have been processed by the **mvsexpt** utility. This file must not be edited or deleted because it is read by subsequent invocations of the **mvsimpt** (pass one) or **mvsexpt** (pass one) utility to filter out users already processed by this file. If this file is deleted and the **mvsimpt** or **mvsexpt** utility is run, entries already processed are reprocessed by the utility with varying results.

  Entries that were not completed successfully are removed from the Processed Entries file and can be found in the RACF Error file. It is important that failed entries be resolved before running **mvsexpt -p2** again because the RACF error file is nulled upon subsequent invocations of **mvsexpt -p2**..

The cross linking process is now complete.

Single Sign-on for OS/390 and DCE

In conjunction with RACF interoperability, DCE provides users the ability to effectively **sign on** (log in) to both OS/390 and DCE in one operation. OS/390-DCE single sign-on allows an OS/390 user who has already been authenticated to RACF to be logged in to DCE. DCE does this automatically when a DCE application is started in an address space and the user is not already logged in to DCE.

Note: Single sign-on is not supported for servers that must log in to DCE. Servers must use DCE interfaces provided in MVS/ESA OpenEdition DCE Release 1.

Preparing for DCE Single Sign-on

Before OS/390-DCE single sign-on can be used by an OS/390 user, the RACF administrator must enroll the user for single sign-on support. To enroll a user, do these steps:

1. Create (or update) a DCE segment for the OS/390 user, as described in “Cross Linking Existing RACF Users who are New DCE Users” on page 400, in “Cross Linking Existing DCE Users who are New RACF Users” on page 405, or in “Cross Linking Existing DCE Users who are Existing RACF Users” on page 407.
2. Be sure that the **AUTOLOGIN** flag in the OS/390 user's DCE segment is set to **YES**. Because the default setting is **NO** (single sign-on is *not* enabled), AUTOLOGIN=YES must be explicit. See “Tailoring Variables for mvsexpt” on page 394 for more information.
3. Have the OS/390 user store his or her current DCE password in the RACF registry using the **storepw** command. The password is also updated in the DCE registry if the **-r** option is specified. Notify users that they must do this before invoking a DCE application, and that each user's principal must be in the security manager (such as RACF) before **storepw** can update the user's DCE registry entry. The **storepw** command is described in the *OS/390 DCE: Command Reference*.

Note: If the DCE security server is running on the local system, it is not necessary to store the password in the RACF registry. In this case, the DCE security server uses the current RACF identity to perform the DCE login without requiring a password.

Automatic DCE Single Sign-on Invocation

After all users requiring single sign-on are enrolled, they can authenticate themselves to OS/390 and run DCE applications. DCE single sign-on is called when a DCE application is run and the user is not already logged in to DCE.

User Control of Automatic DCE Single Sign-on

DCE allows individual users the ability to control whether they have OS/390-DCE single sign-on, after the RACF administrator has set **AUTOLOGIN** to **YES** in the DCE segment for each user. If the RACF administrator has set **AUTOLOGIN** to **NO** or if there is no setting for **AUTOLOGIN** in a user's DCE segment, then that user does *not* have control over automatic single sign-on. In other words, a user may override the **AUTOLOGIN** setting in the DCE segment only if it is set to YES.

The mechanism for overriding the AUTOLOGIN=YES setting is an environment variable called **_EUV_AUTOLOG**, which must be in each user's UNIX System Services shell or environment variables file. The only value that a user can specify for this variable is NO. (Any other variable is ignored.) This environment variable must be set by the user, but if there is no **_EUV_AUTOLOG** environment variable (or it is set to something other than NO), the **AUTOLOGIN** value in the user's DCE segment is used.

For more on how to use DCE environment variables, see Appendix A, “Environment Variables in OS/390 DCE” on page 467.

Chapter 42. Maintaining Policies and Properties

Registry polices 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 are set for the registry as a whole and cannot be set for individual organizations or accounts. Properties regulate such things as the range of numbers that can be used for UNIX IDs and whether encrypted passwords are displayed.

You can set both polices 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.

This chapter first describes policies and then properties.

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.

Standard Policy

Standard policy regulates such things as account and password lifetimes and password format. It can be set for the registry as a whole and for specific organizations. The standard policies you can set are described in the following subsections.

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, incorrect login handling, and expired password handling by attaching ERAs to principals. See Chapter 36, “Creating and Maintaining Principals, Groups, and Organizations” on page 331 for information on how to do this.

Account Lifespan: The account lifespan 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 not valid and must be created again.

You define the account lifespan with the **dcecp acctlife** attribute in the form:

```
acctlife {time| unlimited}
```

The variable *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 using the **dcecp account modify** command to mark the accounts as not valid (**acctvalid** attribute). See Chapter 37, "Creating and Maintaining Accounts" on page 347 for more information.

Password Lifespan:

The password lifespan you set determines 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.

You define the password lifespan as the **dcecp pwdlife** attribute in the form:

pwdlife {*time* **| unlimited}**

The variable *time* is a number that indicates the number of days the password is valid; **unlimited** specifies an unlimited lifespan.

Password Expiration Date: You can also set the exact date passwords expire by using the password expiration date policy (**pwdexpdate** attribute).

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.

You define the password expiration date as the **dcecp pwdexpdate** attribute in the form:

pwdexpdate {*date* **| none}**

The variable *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 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 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 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 Chapter 46, "Accessing Registry Objects" on page 429.

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 subsections.

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 Chapter 46, "Accessing Registry Objects" on page 429 for a general discussion of preauthentication and information on preauthentication administration.

Maximum Ticket Renewable Time:

Note: This feature is not currently used by DCE; any use of this option is unsupported at the present time.

The maximum ticket renewable time (**maxtktrenew** attribute) 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 time, the greater the security of the system. However, because 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 you can set this time for individual accounts 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 requests a ticket to a server, the lifetime granted to the ticket takes into account the maximum ticket lifetime set for both the server and the client. The lifetime 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
```

The variable *hours* is a number that indicates how many 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 you 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 standard policy you set for the registry as a whole differs from the policy 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 28 on page 414 lists the stricter policy for each policy type.

Table 28. 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 of all spaces. It must include characters.
Password consisting of all alphanumerics	The password cannot consist of all alphanumerics. 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” on page 412, 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 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 *shorter* password, the longer length policy is enforced.

Displaying and Setting Standard and Authentication Policies

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 examples 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}
```

Properties

The **dcecp registry modify** command sets properties for the registry as a whole. The properties that you can set are described in the following subsections.

Default Ticket Lifetime Property

The default ticket lifetime is the default lifetime (in hours) for tickets issued to principals in the registry.

You set default ticket lifetimes with the **dcecp deftktlife** attribute in the form:

deftktlife *hours*

hours a number indicating the number of hours in the lifetime.

Hidden Password Property

The hidden password property determines whether encrypted passwords are displayed. You set the hidden password property with the **dcecp hidepwd** attribute in the form:

hidepwd {yes | no}

If you set this property to **yes**, an asterisk (*) is displayed in place of the encrypted password in command output and in files where passwords are displayed. If you set this property to **no**, the hidden password is displayed.

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 Organization 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. 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 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 minutes 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.

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}
```

Chapter 43. Performing Routine Maintenance

This chapter describes Security maintenance procedures that should be performed on a regular basis, such as:

- Adding new users to the registry
- Changing the master key
- Backing up and restoring the database

Adding Accounts

To add new user accounts to the registry, you must have the appropriate permissions to the registry (see Chapter 46, “Accessing Registry Objects” on page 429). After you have the appropriate permissions, you can proceed as follows to add accounts:

1. If the principal to be used in the account does not already exist, run the **dcecp principal create** command to add the principal.
2. Run the **dcecp group create** command to add the group to be used in the account if this group does not already exist.
3. Run the **dcecp organization create** command to add the organization to be used in the account if this organization does not already exist.
4. Finally, run the **dcecp account create** command to add the account.

Or, to do all of these operations using a single command, type:

```
dcecp user create -force
```

When you add new user accounts, and one or more of those users is to be cross linked to RACF, remember to run the RACF interoperability utility, **mvsexpt**, so that the new users will have single sign-on capability and interoperability between RACF and OS/390 DCE. The **mvsimpt** utility creates the **dcecp user create** commands and then processes them to put the principal names into the registry. For more information, see “Cross Linking Existing DCE Users who are New RACF Users” on page 405.

Also, for users who are to be enabled for single sign-on, remind them that they must each use the OS/390 DCE **storepw** command before invoking a DCE application from the OS/390 system. The **storepw** command must also be run any time the user changes his or her password. For more information on the **storepw** command, see the *OS/390 DCE: Command Reference*.

Changing the Registry's Master Key

All passwords stored in a 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 reencrypt all passwords with the new master key. Each replica (master and slave) 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 reencrypts all the passwords for the replica **art_server_1**:

```
registry modify /.../giverny.com/subsys/dce/sec/art_server_1 -key
```

Validating the Authenticity of the DCE Security Service

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.

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.

The following example illustrates a **secval ping** operation to the **secval** process on remote host **charon**:

```
dcecp> secval ping /.:/hosts/charon
1
dcecp>
```

Backing Up and Restoring the Registry Database

Use the exact procedures described here to back up the registry database to prevent backups from arriving at the master during the backup.

Only the master replica database and its master key file need to be backed up. Use the procedures that are described in the following subsections 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.

Procedures for Backing Up the Registry Database

To run the backup procedures, ensure that you are logged into DCE through an administrative account. The back-up steps are as follows:

1. Enter the **registry disable** command to set the master replica to the maintenance state. The following command sets the master 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 master 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 is done, take the master replica out of maintenance state, as follows:

   ```
   dcecp> registry enable  /.../giverny.com/subsys/dce/sec/master
   dcecp>
   ```

   The OS/390 Security Server resumes accepting updates.

Note that the previous examples supplied the name of the registry master 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 master registry of the machine's default cell. See “Setting the _s(sec) Variable” on page 422 for more information.

Procedure for 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” on page 423.

To restore the registry database to a machine, perform the following steps:

1. Log in as **root** at the master 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:

   ```
   $ /bin/secd -restore_master &
   ```

   This command will start **secd** and cause the master to mark all slaves to be reinitialized.

5. This is one of the few times on an OS/390 system that the registry server is started in the shell environment. While the registry server is running in this mode, do not enter any other DCE commands from this logon. Stop **secd** after it has completed its restoration and has reinitialized the other security replicas in the cell. This is done by using the **kill** command.
6. Start the registry server by typing the OS/390 operator command:

   ```
   modify dcekern,start secd
   ```

   This starts **secd** as a DCEKERN process where it usually runs.

7. 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.

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 master replica. 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 master replica (named **slave_3**) in the cell **giverny.com**:

```
dcecp> set _s(sec) /.../giverny.com/subsys/dce/sec/slave_3
dcecp>
```

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 appears on the replica list of the current default replica (that is, its cell-relative name; for example, **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 master replica and thus require binding to the master. If you run a command that acts only on the master and the master is not the default replica, **dcecp** automatically attempts to bind to the master 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 master registry. The master 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.

Chapter 44. Handling Network Reconfigurations

This chapter describes the procedures to handle network reconfigurations that change the locations of registry replicas. Specifically, this chapter covers:

- Changing the master registry site
- Removing a node from the network
- Handling network address changes

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

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 master registry site is to use the **dcecp registry designate** command to reverse the roles of the master server and a slave server. In other words, make the master the slave and the slave the master.

When you run 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 slaves.
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 master registry to another machine when the registry servers at the master and slave 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 error or because the master has gone down temporarily.

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 DCECONF command to configure a slave machine.
2. Enter the **set** command to set default replica to the current master replica. In the example below, the master replica is set to the replica named **master** in the cell **giverny.com**:

   ```
   dcecp> set _s(sec) /.../giverny.com/subsys/dce/sec/master
   dcecp>
   ```

3. Enter the **registry designate** command to reverse the roles of the master and slave. When you enter this command, you supply as an argument the name of the replica to be made the new master. 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.

Removing a Server Machine from the Network

If you are planning to remove a machine that runs a slave 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 master server, you must change the master server site, as described previously, before you remove the node.

Use the **dcecp registry delete** command to delete a slave replica. When you run this command, the master 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

The following sample command deletes the slave replica named **/.../giverny.com/subsys/dce/sec/art_1**

```
dcecp> registry delete /.../giverny.com/subsys/dce/sec/art_1
dcecp>
```

When you enter this command, **dcecp** binds to the master replica that is in the current cell, if necessary. Then the master replica instructs the slave replica to delete itself.

Verify that the slave is deleted by issuing the **dcecp registry catalog** command. When the master has received the request to delete the slave, the slave appears on the replica list as marked for deletion. When the replica has actually been deleted, it no longer appears on the list.

| If the replica that you are removing is a security server, and it is being removed from a node that will
| otherwise remain intact, you must manually change the **pe_site** file in **/opt/dcelocal/etc/security**. If you
| do not change this file, important DCE components may not be able to find the new security master
| replica on the network. See “Updating the pe_site File” for more information.

Handling Network Address Changes

When **secd** starts, master and slave replicas can detect address changes and can perform the necessary updates to the master'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 master and a slave replica change simultaneously, your intervention is required. This subsection describes how to update the **pe_site** file and how to handle simultaneous address changes.

Updating the pe_site File

Whenever the master's or a slave'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 master replica, the file contains the cell name and the name of the master. For slave replicas, the file contains the cell name, the name of the master replica, and the name of the replica itself.

If the master replica address changes, update the **pe_site** file on every node in the cell that runs a security server (including the master) with the new address for the master. If a slave address changes, update only that slave's **pe_site** file to reflect its changed address.

Handling Simultaneous Address Changes

If an address change occurs simultaneously for the master replica and a slave replica, the master and slave 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 master or slave) 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 master site and the slave 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 master and a slave, the only way to restore communication between the master and slave is to delete the slave, then recreate it. Delete the slave using one of the following methods, depending on your circumstances:

- If you anticipate a simultaneous address change, while the master and slave are still communicating, use the **dcecp set** command to bind to the master and then the **dcecp registry delete** command to delete the slave replica.
- If the **secd** is running at the master and slave sites, but the master and slave are not communicating, first use the **dcecp set** command to bind to the slave and then the **registry delete -only** command to destroy the slave. Then use **dcecp set** to bind to the master and **registry delete -force** command to remove the replica list entry for the slave.

Chapter 45. 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.

This chapter describes how to **adopt** these objects which are not associated with any principal or group.

What are Orphans?

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.

Solving the Problem of Orphans

To solve this problem, 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.

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 manner in which it is created, a principal, group, or organization created by these commands is no different from any other principal, group, or organization. The following examples show how to use this option to create a principal, group, or organization to adopt an orphaned registry object.

To create a principal associated with 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]
```

where:

name	The principal's, group's, or organization's primary name.
uuid	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	The principal's, group's,or organization's full name.
UNIX_number	For cell principals *only,* 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 embedded in the principal's UUID and cannot be specified here.
object_creation_quota	For principals *only*, 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 property 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 do not succeed, it could be because the embedded UNIX number is not valid 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 numbers to include the UNIX number embedded in the UUID and then try again to adopt the object. See Chapter 42, “Maintaining Policies and Properties” on page 411 for information on setting the valid range of UNIX numbers.

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]
```

Chapter 46. Accessing Registry Objects

This chapter describes the permissions that apply to objects in the registry. Because the permissions granted are based on the way the registry database is structured, this chapter first briefly describes the structure of the registry database. It then describes the permissions for each type of object in the registry database, the registry ACL managers, and the initial registry ACLs.

Both the **dcecp** command and the **acl_edit** command have functions for creating, modifying, and deleting ACL entries for registry objects. See those sections in the *OS/390 DCE: Command Reference* for each command for descriptions of the operations they perform on ACL entries.

The Registry Database

The registry is structured into the following main directories:

- **The Principal Directory** contains information about principals.
- **The Group Directory** contains information about groups.
- **The Org Directory** contains information about organizations.

In addition to the directories, the registry contains the **policy** object, the **replist** object, and the **xattrschema** object, all of which are created when the registry is created during machine configuration. The **policy** object contains information that applies to the registry properties and policies, and to organization policies. The **replist** object contains information about the replicas in your cell. And, the **xattrschema** object contains information about extended registry attributes. You can change **policy** and **replica** information at any time by using the **dcecp registry** commands. The **xattrschema** object is changed 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 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 granted to objects in the registry depend on where the object fits in the structure of the registry database. Figure 65 illustrates the registry database. The square boxes represent container objects (directories). The ovals represent simple objects. Figure 65 shows only the top level Principal, Group, and Org directories. Your registry can have subdirectories if you create them.

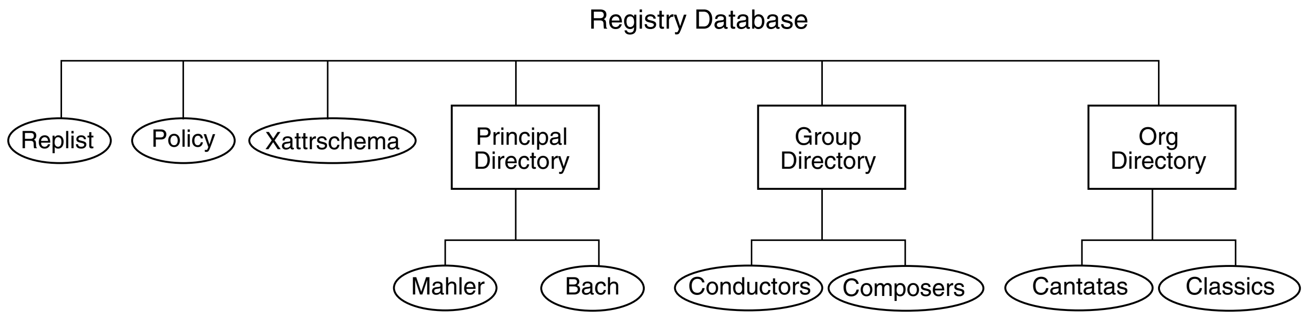


Figure 65. Registry Database Structure

Registry Permissions

Table 29 lists the permissions that can be granted for the object types found in the registry.

Table 29. Permissions for Registry Objects

Permission	Meaning
A	Run commands that act on replicas (**sec_admin**).
a	Modifies authentication information.
c	Changes 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	Changes 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	Changes management information.
M	Adds and deletes members from a group or organization. To add a member to a group, you must also have **g** permission for the principal to be added.
n	Changes the name of a directory, a principal, a group, or an organization.
u	Changes user information.
r	Views management, authentication, and user information.
t	Tests the group or organization membership of a named principal.

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 information kept for each object type is summarized in the subsections that follow.

Management Information:

Management information includes:

- **For registry policies and properties**
  - The account lifespan
  - The password minimum length
  - The password lifespan
  - Whether passwords can contain spaces
  - Whether passwords can consist of all nonalphanumeric characters
  - The password expiration date
  - The minimum ticket lifetime
  - The default ticket lifetime
  - A number that defines the lowest UNIX ID 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**
  - The account, group, and organization names
  - Text string showing the full name of the principal
  - Object creation quota for the principal
  - Whether the principal can change primary names to aliases and aliases to primary names
  - User Identifier (UID) of the principal
  - Unique User Identifier (UUID) of the principal
  - 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 client or as a server
- **For groups**
  - Primary name of the group
  - Text string showing the full name of the group
  - Whether the group's primary name can be changed to an alias and its aliases to its primary name
  - Group Identifier (GID) for the group
  - The project list inclusion property
  - Unique User Identifier (UUID) of the group
- **For organizations**
  - Primary name of the organization
  - Whether the organization's primary name can be changed to an alias and its aliases to its primary name
  - Text string showing the full name of the organization
  - Organization Identifier (ORGID) for the organization
  - Unique User Identifier (UUID) of the organization
  - The account lifespan
  - The password minimum length
  - The password lifespan
  - The password expiration date
  - Whether passwords can contain spaces
  - Whether passwords can consist of all nonalphanumeric characters
- **For the xattrschema object**
  - Whether the xattrschema can be modified

Authentication Information: Authentication information includes:

- **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 the principal's account was last changed (Good Since Date)
  - 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
  - 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

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 to Create Principals, Groups, or Organizations

Figure 66 shows the permission required to create principals, groups, or organizations.

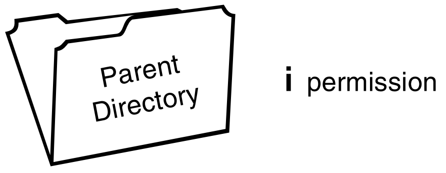


Figure 66. Permission Required To Create Principals, Groups, or Organizations

To create a principal, group, or organization, you must have **i** permission on the directory in which you create the principal, group, or organization. For example, to create the principal **preludes/villa/lobos**, you must have **i** permission on **villa**.

Permissions to Delete Principals, Group, or Organizations

Figure 67 on page 433 shows the permissions required to delete principals, groups, or organizations.

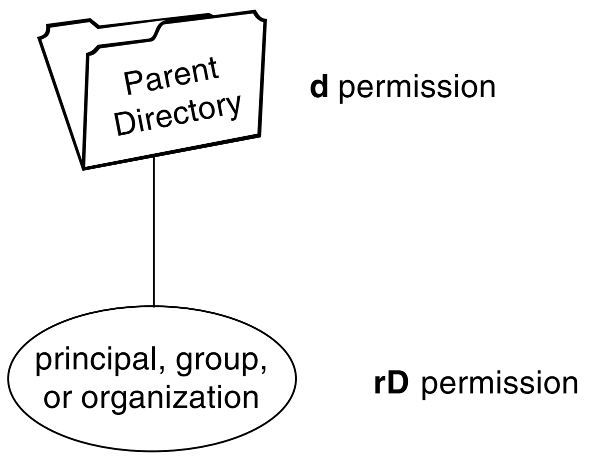


Figure 67. Permissions Required To Delete Principals, Groups, or Organizations

To delete principals, groups, or organizations, you must have the following permissions:

- **d** permission on the directory in which the principal to be deleted exists
- **rD** permission on the principal, group, or organization to be deleted

For example, to delete the principal **preludes/villa/lobos**, you must have **d** permission for the **preludes/villa** directory, and **rD** permission for the principal **preludes/villa/lobos**.

Permissions to Add Accounts

When you add accounts, the **dcecp**, or **rgy_edit**, command adds the principal to the group or organization named in the account, if the principal is not already a member of the group or organization. For this reason, the permissions required to add an account may include the permissions 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 the 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” on page 434.)
- 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” on page 434.)
- The permissions required to add an account and add the principal only to the group 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” on page 435.)
- The permissions 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” on page 436.)

Adding an Account and the Account Principal to the Group and Organization:

Figure 68 on page 434 shows the permissions required to add an account and the account principals to the group or organization.

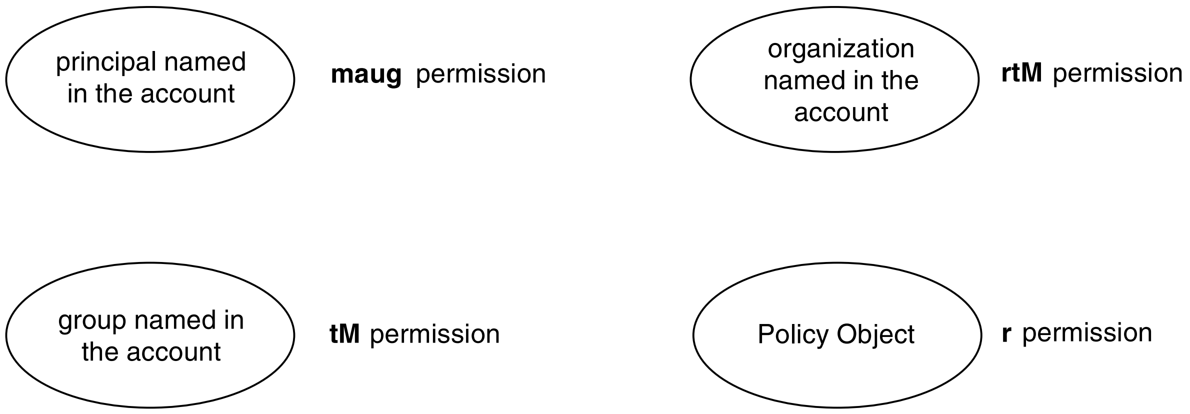

principal named in the account
maug permission
organization named in the account
rtM permission
group named in the account
tM permission
Policy Object
r permission

Figure 68. Permissions Required to Add 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:

- **maug** permissions on the account's principal
- **tM** permissions on the group named in the account
- **rtM** permissions on the organization named in the account
- **r** permission on the registry **policy** object

For example, to create an account for the principal **preludes/villa/lobos** associated with the group **composers** and the organization **pianists**, you must have:

- **maug** permissions on **preludes/villa/lobos**
- **tM** permissions on the group **composers**
- **rtM** permissions on the organization **pianists**
- **r** permission on the registry **policy** object

Adding an Account for which the Principal is Already a Member of the Group and Organization: Figure 69 on page 435 shows the permissions required to add an account for which the principal is already a member of the group and organization.

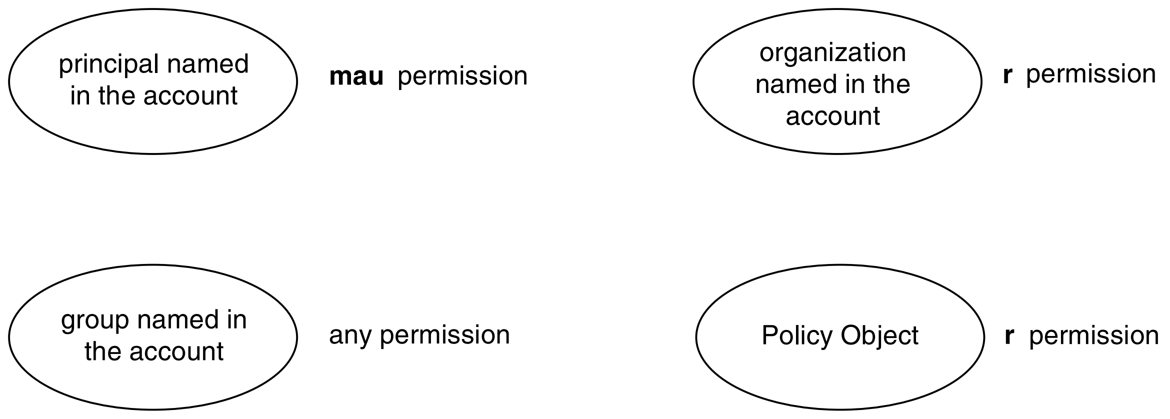

principal named in the account
mau permission
organization named in the account
r permission
group named in the account
any permission
Policy Object
r permission

Figure 69. 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:

- **mau** permissions on the account principal
- At least one permission of any kind on the group named in the account
- **r** permission on the organization named in the account
- **r** permission on the registry **policy** object

For example, to create an account for the principal **preludes/villa/lobos** associated with the group **composers** and the organization **pianists**, you must have:

- **mau** permissions on **preludes/villa/lobos**
- At least one permission of any kind on the group **composers**
- **r** permission on the organization **pianists**
- **r** permission on the registry **policy** object

Adding an Account and the Principal to the Group Only: Figure 70 shows the permissions required to add an account and the principal to the group only.

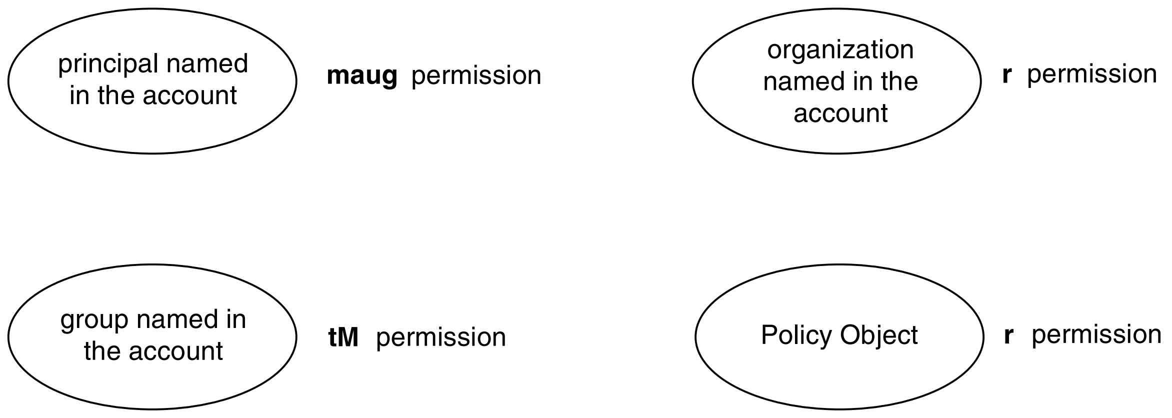

principal named in the account
maug permission
organization named in the account
r permission
group named in the account
tM permission
Policy Object
r permission

Figure 70. Permissions to Add 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:

- **maug** permissions on the account's principal
- **tM** permissions on the group named in the account
- **r** permission on the organization named in the account
- **r** permission on the registry **policy** object

Adding an Account and the Principal to the Organization Only: Figure 71 shows the permissions required to add an account and the principal to the organization only.

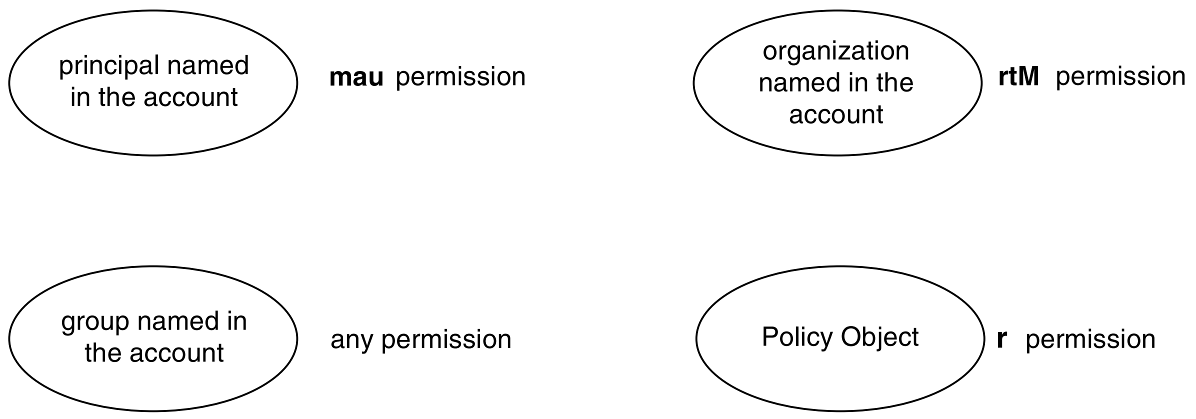

principal named in the account
mau permission
organization named in the account
rtM permission
group named in the account
any permission
Policy Object
r permission

Figure 71. Permissions to Add an Account and the Principal to the Organization Only

To add an account and add the account's principal to the organization (the principal is already a member of the group named in the account), you must have:

- **mau** permissions on the account's principal
- At least one permission of any type on the group named in the account
- **rtM** permissions on the organization named in the account
- **r** permission on the registry **policy** object

Permissions to Delete Accounts

Figure 72 shows the permissions required to delete accounts.

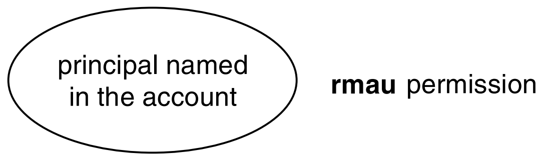


Figure 72. Permissions Required to Delete Accounts

To delete accounts, you must have **rmau** permissions for the principal named in the account. For example, to delete the account for the principal named **preludes/villa/lobos**, you must have **rmau** permissions for **preludes/villa/lobos**.

Permissions to Add Members to Groups

Figure 73 on page 437 shows the permissions required to add members to groups.

principal to be added to the group
rg permission
group to which the principal is to be added
rM permission

Figure 73. Permissions Required to Add Members to Groups

To add members to groups, you must have:

- **rM** permissions on the group to which the principal is being added
- **rg** permissions on the principal to be added

For example, to add the principal **preludes/villa/lobos** to the group **composers**, you must have:

- **rM** permissions on the group **composers**
- **rg** permissions on the principal **lobos**

Permissions to Add Members to Organizations

Figure 74 shows the permissions required to add members to organizations.

principal to be added to the orgranization
r permission
org. to which the principal is to be added
rM permission

Figure 74. Permissions Required to Add Members to Organizations

To add members to organizations, you must have:

- **rM** permissions on the organization to which the principal is being added
- **r** permissions on the principal to be added

For example, to add the principal **preludes/villa/lobos** to the organization **pianists**, you must have:

- **rM** permissions on the organization **pianists**
- **r** permission on the principal **lobos**

Permissions to Delete Members from Groups or Organizations

Figure 75 on page 438 shows the permissions required to delete members from groups or organizations.

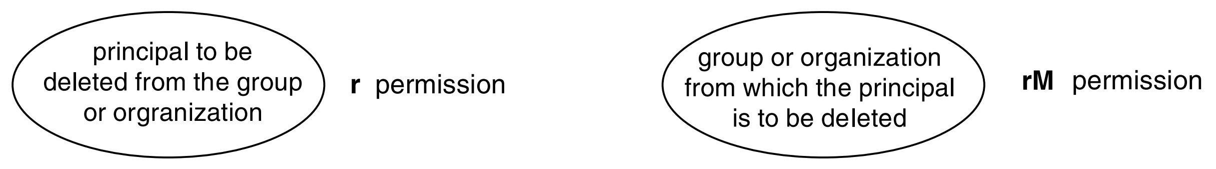

principal to be deleted from the group or orgranization
r permission
group or organization from which the principal is to be deleted
rM permission

Figure 75. Permissions to Delete Members from Groups or Organizations

To delete members from a group or organization, you need **rM** permissions on the group or organization from which the principal is being deleted and **r** permission on the principal being deleted.

For example, to delete the principal **preludes/villa/lobos** from the group **composers**, you must have:

- **rM** permissions on the group **composers**
- **r** permission on the principal **lobos**

Permissions to Change a Principal's, Group's, or Organization's Full Name

Figure 76 shows the permissions required to change a principal's, a group's, or an organization's full name.

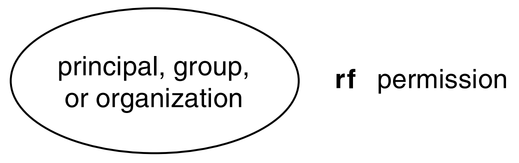


Figure 76. Permissions Required to Change a Principal's, Group's, or Organization's Full Name

To change a principal's, group's, or organization's full name, you must have **rf** permissions for the principal, group, or organization for which you are making the change.

Permissions to Change Management Information for Principals, Groups, or Organizations

Figure 77 shows the permissions required to change management information for principals, groups, or organizations.

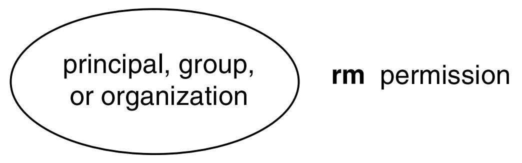


Figure 77. Permissions Required to Change Management Information for Principals, Groups, or Organizations

To change management information for a principal, a group, or an organization, you must have **rm** permissions for the object for which you are changing management information.

Permissions to Change Management, Authentication, and User Information (Except Passwords) for Accounts

Figure 78 on page 439 shows the permissions required to change management, authentication, and user information (except passwords) for accounts.

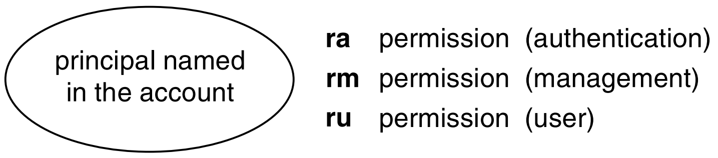

principal named in the account
ra permission (authentication)
rm permission (management)
ru permission (user)

Figure 78. Permissions Required to Change Management, Authentication, and User Information (Except Passwords) for Accounts

To change all management, authentication, and user information (except passwords) for accounts, you must have the following permissions for the principal named in the account:

- **ra** permission to change authentication information
- **rm** permission to change management information
- **ru** permission to change user information

Permissions to Change Passwords for Accounts

Figure 79 shows the permissions required to change passwords for accounts.

principal named in the account
ru permission
Policy Object
r permission

Figure 79. Permissions Required to Change Passwords for Accounts

To change passwords for accounts, you must have these permissions for the principal named in the account:

- **ru** permissions on the account's principal
- **r** permission on the registry **policy** object

Permissions to Change Authentication and Management Information for Registry Policies and Properties

Figure 80 on page 440 shows the permissions required to change authentication and management information for registry policies and properties.

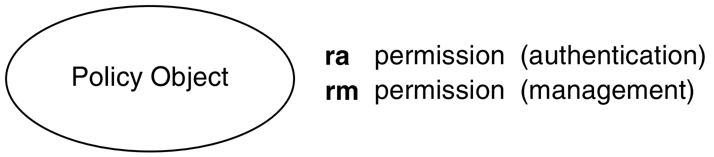

Policy Object
ra permission (authentication)
rm permission (management)

Figure 80. Permissions Required to Change Authentication and Management Information for Registry Policies and Properties

To change management or authentication information for the registry using the **dcecp registry modify** command or the **rgy_edit prop**, **po**, and **auth** commands, you must have **ra** permissions (to change authentication information) or **rm** permissions (to change management information) for the registry **policy** object.

Permissions to Run Commands that Act on Replicas

Figure 81 shows the permissions required to run commands that act on replicas.

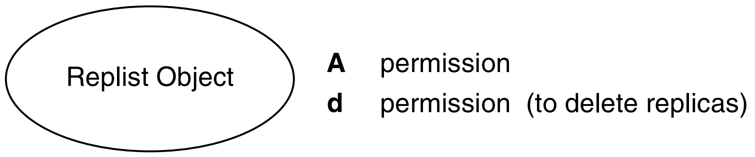


Figure 81. Permissions Required to Run Commands that Act on Replicas

To run any of the commands that act on replicas, you must have these permissions on the **replist** object:

- **A** permission to run all commands except for those that display replica information, which require no permissions on the **replist** object.
- **d** permission to run the commands that delete replicas.

Permissions to Create Extended Registry Attribute Types

Figure 82 shows the permission required to create extended registry attribute types.

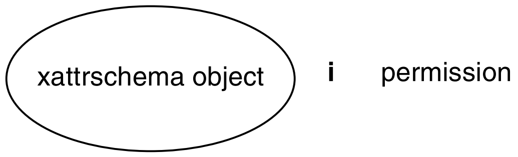


Figure 82. Permissions Required To Create Extended Registry Attribute Types

To create an extended registry attribute type in the registry schema, you must have **i** permission on the xattrschema object.

Permissions to Delete Extended Attribute Types

Figure 83 on page 441 shows the permissions required to delete extended attribute types.

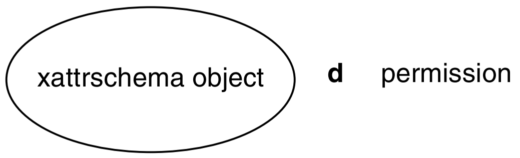


Figure 83. Permissions Required To Delete Extended Registry Attribute Types

To delete extended attribute types, you must have **d** permission on the xattrschema object.

Permissions to View Extended Registry Attribute Types: Figure 84 shows the permission required to view one or more ERAs in the registry's schema database (with the **dcecp schema show** command).

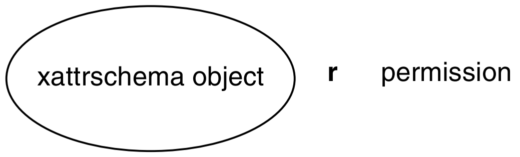


Figure 84. Permissions Required To View Extended Registry Attributes

To view extended registry attribute types, you must have **r** permission on the xattrschema object.

Permissions to Modify Extended Registry Attribute Types

Figure 85 shows the permission required to modify extended registry attribute types.

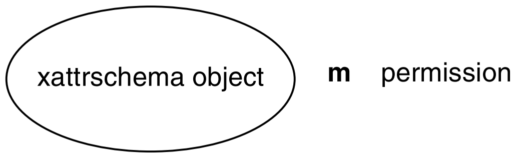


Figure 85. Permissions Required To Modify Extended Registry Attribute Types

To modify extended registry attribute types, you must have **m** permission on the xattrschema object.

Permission to Change ACLs on Registry Objects

Figure 86 shows the permissions required to change ACLs on registry objects.

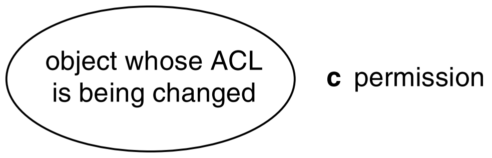


Figure 86. Permission Required to Change ACLs on Registry Objects

To change ACLs on registry objects, you must have **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 be initialized and to function properly, slave replicas must have **i**, **m**, and **I** permissions to the **replist** object (**/.:/sec/replist**). A slave server runs under the identity of the machine on which it runs. A machine name is the local host principal name in the form:

host/*hostname***/self**

The required ACL entry is added during the initial configuration of the cell's security server and when the initial configuration program is used to create new slave replicas. The entry has the form:

user:host/*hostname***/self:imI**

Registry ACL Manager

The registry ACL Manager consists of five manager types to handle different ACL semantics 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 different from those of a **principal** object, there is a separate Group ACL Manager type that controls the ACLs on **group** objects.

Not all permissions nor all ACL entry types are valid for each ACL Manager. Table 30 summarizes the permissions that are valid and those that are not valid, as well as the ACL entry types that are not valid for each ACL Manager.

Table 30. ACL Managers, Valid Permissions, and ACL Entry Types.

Manager Type	Valid Permissions	ACL Entry Types that are Not Valid
dir (controls **directory** objects)	rcidDn	user_obj group_obj
policy (controls the **policy** object)	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 replica lists)	cidmIA	user_obj group_obj
xattrschema (controls extended registry attribute types)	rcidm	user_obj group_obj

Initial Registry ACLs

When the registry database is created on the host where the security server is running, 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 the chapter on setting up the registry in *OS/390 DCE: Configuring and Getting Started* for more information on **sec_create_db** and the registry creator.)

Note: Although the OS/390 Security Server is not part of OS/390 DCE Base Services, it *is* available as an optional feature of OS/390. The **sec_create_db** command is included in this feature. The management interfaces to OS/390 Security Server (**dcecp**, Registry Editor, and ACL Editor) are part of the OS/390 DCE Base Services.

The initial ACLs created when the registry database is created are described in the following list. In the list, *rgy_creator* signifies the principal named as the registry creator.

- For **principal** Objects

  -
    ```
    unauthenticated:r--------
    user_obj:r---f--ug
    user:rgy_creator:rcDnFmaug
    other_obj:r-------g
    any_other:r--------
    ```
- For **group** Objects

  -
    ```
    unauthenticated:r-t-----
    user:rgy_creator:rctDnfmM
    group_obj:r-t-----
    other_obj:r-t-----
    any_other:r-t-----
    ```
- For **org** Objects

  -
    ```
    unauthenticated:r-t-----
    user:rgy_creator:rctDnfmM
    other_obj:r-t-----
    any_other:r-t-----
    ```
- For the **policy** Object

  -
    ```
    unauthenticated:r----
    user:rgy_creator:rcma
    other_obj:r----
    any_other:r----
    ```
- For **directory** objects

  -
    ```
    unauthenticated:r-----
    user:rgy_creator:rcidDn
    other_obj:r-----
    any_other:r-----
    ```
- For the **replist** Object
  - `user:cell_admin:cidmA-`
- For the **xattrschema** Object

–
```
unauthenticated:r-----
user:cell_admin:rcidm
other_obj:r-----
any_other:r-----
```

Note: Your platform's configuration tool may update these initial ACLs.

Chapter 47. 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.

Audit, a key component of DCE, is provided by the DCE Audit Service.

This chapter provides an introduction to the DCE Audit Service.

Features of the DCE Audit Service

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 system console.

Components of DCE Audit Service

The DCE Audit Service has three basic components:

Application Programming Interfaces (APIs)	Provide the functions that 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 control program	The DCE Audit Service's management interface to the Audit daemon. As an administrator, you can use it to specify how the Audit daemon will filter the recording of audit events.

DCE Audit Service Concepts

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 "Code Point" on page 446), are already in place for 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 chapter 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 require auditing. For example, if a bank server offers the cash withdrawal function **acct_withdraw()**, this function can be assumed to be an auditable event and 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** section of Appendix H, "DCE Security Administration Files" on page 553. Code points and their associated events for the DCE Distributed Time Service are documented in the **dts_audit_events** section of Appendix H, "DCE Security Administration Files" on page 553. Code points and their associated events for the DCE Audit Service are documented in the **aud_audit_events** section of Appendix H, "DCE Security Administration Files" on page 553.

Audit Event

An audit event is any event that an audit client wishes 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.

For a description of audit event files available in OS/390 DCE and how to use them, see Appendix H, "DCE Security Administration Files" on page 553.

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 "Administration and Programming in DCE Audit" on page 451.

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 DCE 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.

Note: In OS/390 UNIX System Services DCE, all audit event class files must be in code page IBM-1047.

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 is an example of an event class name:

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

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 event-id. The set-id corresponds to a set of event classes and is assigned by OSF to an organization or vendor. The event-id identifies an event class within the set of event classes. The organization or vendor manages the issuance of the event-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 event-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 event-id. This format allocates 7 bits to the set-id and 24 bits to the event-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 event-id.

Format C can be used by small organizations that need less than 8 bits for the event-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 1110 in the most significant bits (that is, 0xE0000000 to 0xEFFFFFFF) are reserved to be used locally within a cell.

The event class number formats are illustrated in the following figure.

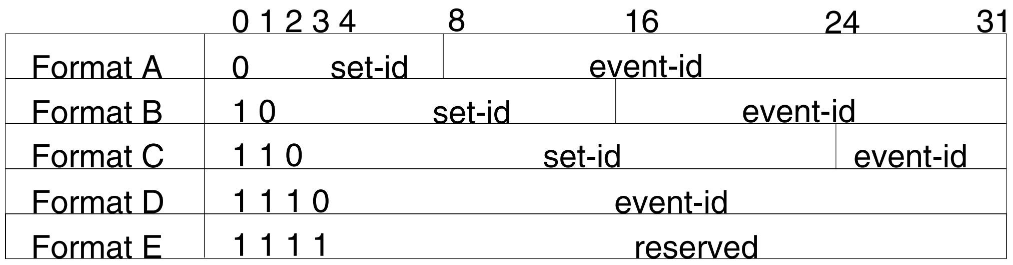


Figure 87. Event Class Number formats

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

Filters

After 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 (**dcecp**).

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 do not succeed 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).

Example of Filter Guides: The following is an example of a filter with two guides:

```
filter type: foreign_principal
key: /.../dcecell1.endicott.ibm.com/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 will be logged for any event in event class **Confidential**, if the user is the foreign principal **/.../dcecell1.endicott.ibm.com/foo** and the event failed because of access denial. Guide 2 specifies that an audit record will not only be logged but also be displayed on the system console for any event in event class **Restricted**, for the same user and event outcome.

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.

Example of Using Filter Rules: 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), and regardless of 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). Thus, these same events will cause an audit record to be logged and also raise an alarm.

Non-overridable 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* non-overridable 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.

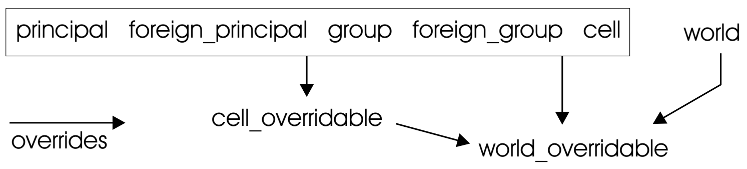


Figure 88. Override Relations between Filter Types

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 are not overridable.

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 called with a name for the trail file, this name becomes the pathname to the local audit trail file and all audit records are sent to that file.

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.

A banking server example illustrates this scenario.

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.

   For example, a banking server program can have the following functions: **acct_open()**, **acct_close()**, **acct_withdraw()**, **acct_deposit()**, and **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 */
     .....
     .....
   ```

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.

   For example:

   ```
      /* 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
   ```

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. For example:

   ```
   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);
      ....
   ```

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.

   Following is an example of how these APIs are used 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);
 .....
 .....
 .....
```

5. Closes the audit trail file when the server shuts down, using the **dce_aud_close** API in the main server routine. For example:

```
  .......
dce_aud_close(audit_trail, &status);
  .......
```

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. In our example, these code points were assigned the following event numbers:

acct_open()	0x01000000
acct_close()	0x01000001
acct_withdraw()	0x01000002
acct_deposit()	0x01000003
acct_transfer()	0x01000004

(Note that event numbers should be completely sequential. That is, no missing members in the sequence is allowed.)

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 0xC1000006. Event class account_balance_operations is assigned the event class number 0xC1000007.

   To create the event classes, the administrator creates and edits two files, one for each event class. The name of each of these files will be the same as the event class that each represents. Each file will contain the numbers of the events in each event class.

   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 = 0xC1000006

# Event number of acct_open()
0xC1000000

# Event number of acct_close()
0xC1000001
```

   The file with the name **account_balance_operations** is edited as follows:

```
# Event class number of account_balance_operations
ECN = 0xC1000007

# 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.

3. The administrator decides to create two filters, one for all users within the cell (for the cell **/.../dcecell1.endicott.ibm.com**), and the other for all other users.

   The filter for all users within the cell has the following guides:

   - 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:

```
dcecp> audfilter create {cell /.../dcecell1.endicott.ibm.com} -attribute \
 {account_balance_operations denial log}
```

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 system console.

Following is the **dcecp** command for creating this filter:

```
dcecp> audfilter create world -attribute \
{{account_balance_operations account_creation_operations} {success failure denial} {alarm log}}
```

Chapter 48, “DCE Audit Service Administrative Tasks” on page 457 provides detailed information about the DCE control program's **audfilter create** command.

Chapter 48. DCE Audit Service Administrative Tasks

This chapter describes the following administrative tasks that are performed for the DCE Audit Service:

- Setting the DCE Audit environment variables.
- Starting (and stopping) the DCE Audit daemon.
- Controlling access to the DCE Audit daemon.
- Creating and maintaining event classes to logically group a set of audit events. Event classes are created by editing event class files.
- Creating and maintaining filters that set the criteria for recording audit events in an audit trail file.
- Enabling and disabling the audit logging service of the DCE Audit daemon.
- Modifying and querying the attributes of the DCE Audit daemon.
- Controlling and displaying the audit trail file.

All of the examples this chapter gives for audit tasks use the DCE control program (**dcecp**).

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.

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, because 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.

Before the Audit daemon can be started, it must be configured. See the *OS/390 DCE: Configuring and Getting Started* for details on configuration. After the Audit daemon is configured, it starts when DCEKERN is started.

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.

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 central 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/hostname/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. You can create a DCE Security group that consists of the servers on the host that are authorized to generate audit records (for example, the 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}
```

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. For example, for the administrator's group **hosts/machine1/audit-admin**:

```
dcecp> acl modify /.:/hosts/machine1/audit-server -add {group hosts/machine1/audit-admin rwc}
```

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 in Defining an Event Class

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 in 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.

Note: In OS/390 UNIX System Services DCE, all audit event class files must be in code page IBM-1047.

In the event class file, empty lines are ignored and comments are designated by the number sign (#) before the comment text.

Example Event Class File

Following is an example of an event class file named **ec_local_cell_critical_events**.

```
ECN = 0xC0000005

# Server Event Number Prefixes
# 0x00000100 Security Service Events
# 0x00000200 Time Service Events
# 0x00000300 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 registry service)
0x00000127
# evt_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
```

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.

How To Create 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 **-attribute** 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**.

In order to run the **audfilter create** command, you must have write (**w**) permission to the Audit daemon's ACL.

How To Modify Filters

You can modify an existing audit filter by adding or deleting one or more of the filter's guides. 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.

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.

How To Delete Filters

You can delete one or more of the audit filters for a DCE client by using the **audfilter delete** command. 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**.

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 run this command, you must have write (**w**) permission to the Audit daemon's ACL.

Default Filters

During the configuration of the Audit daemon, 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), the server's binding information is stored 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 may 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 updates.
```

indicates that the Audit daemon is unable to inform the Audit client of filter updates.

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.

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 OS/390 Security Server and the time server use their own audit trail files and are not affected by use of enable or disable commands.

The **aud stop** command stops the Audit daemon.

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 will overwrite 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. The change is effective only until either the audit daemon stops or until another **aud modify** command is entered.

Controlling and Displaying Audit Trails

The Audit daemon logs audit records sent from audit clients into the central audit trail file. The audit trail name is the name specified during the configuration of the Audit daemon.

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 audit trail file or a local audit trail file.

For example, 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 ---
```

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 megabytes (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.

If you wish to change the size of the audit trail file, you must 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.

For example, if you wish to use a trail file size of 5 MB, the value of **DCEAUDITTRAILSIZE** should be as follows:

```
DCEAUDITTRAILSIZE 5000000
```

The DCEAUDITTRAILSIZE environment variable may also be used to override the audit trail size. This variable is set during the configuration of the Audit daemon.

If for any reason you desire 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 backup 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.

The following example shows how the **aud modify** command causes 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**.

Chapter 49. Hardware Cryptography in DCE

OS/390 DCE is designed to take advantage of the encryption and decryption function in the new generation of System/390® processors.

DCE uses several of these functions for internal system encryption and decryption and for user data privacy. This support is provided by the combination of the Integrated Cryptographic Feature (ICRF) on the processor and the Integrated Cryptographic Service Facility/MVS (ICSF/MVS) software product.

If ICSF/MVS is installed on your OS/390 system, you must authorize the user IDs of users running DCE to access the RACF-controlled ISCF/MVS cryptographic keys and services. This can be done one user ID at a time or on a group basis. See the section on “Controlling Who Can Use Cryptographic Keys and Services” in the *ICSF Administrator's Guide* for more information.

Appendix A. Environment Variables in OS/390 DCE

Environment variables affect the behavior of the OS/390 DCE components. For example, the starting point of searching for a compatible server can be set by defining the environment variable, RPC_DEFAULT_ENTRY. These variables may have different values for each user environment.

In OS/390 DCE, a file called the **envar** file can be created to contain the declarations of the environment variables. You can also choose to override the settings in the **envar** file by setting these variables in the shell, TSO, or batch environments. In the OS/390 shell, environment variables are set using shell commands. In TSO, environment variables are set through a runtime option in the CALL statement. In batch, environment variables are also set through a runtime option in the EXEC statement.

This appendix describes the OS/390 DCE environment variables, and tells you how to set them in the different OS/390 environments.

Table of Environment Variables

The table below provides an extensive list of OS/390 DCE environment variables, including their possible values.

Note: Setting these environment variables is optional. If a value is not given, the default value is used.

Important

In most cases, the default values of these variables are already the correct settings for the user environment. Setting these environment variables is optional. The user can choose not to set these variables, using the default values. The descriptions in this section describe how the user can set these variables, only if it is necessary to do so.

Table 31 (Page 1 of 17). OS/390 DCE Environment Variables

Name	Description
POSIX Environment Variables Used by DCE	
LANG	A POSIX environment variable used by DCE to specify the name of the default locale. This locale is used unless one of the following environment variables is specified: LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, LC_SYNTAX, LC_TIME, LC_TOD. The LANG environment variable may be used in the search path for the message catalogs as defined by the NLSPATH environment variable.
NLSPATH	A POSIX environment variable used by DCE that defines the directory structure where the message catalogs exist.
TZ	Sets the time zone value used by DTS. This variable denotes the time zone in which the user, process, or machine runs. It must be a POSIX TZ value. The default is GMT0.
dcecp ***Controls***	
DCECP_ERROR	Controls whether **dcecp** displays additional error information when a **dcecp** error occurs. **ON** displays the error information. Any other value turns it off. It is recommended that this variable be set to ON when you are running **dcecp** commands in batch. This ensures that any failing command is displayed along with the error messages.

Table 31 (Page 2 of 17). OS/390 DCE Environment Variables

Name	Description
HOME	Specifies the home directory. Needed to determine the location of any files specified with the tilde (˜) expansion. In particular, when **dcecp** is started in interactive mode, HOME is used to locate the **.dcecprc** initialization file. An error message is issued if HOME is not set.
PATH	Specifies the directories to check for an executable file for a command that is not a **dcecp** or Tcl command. It is also used for resolving a command in which the object is abbreviated. An error message is issued if PATH is not set.
TCL_LIBRARY	Specifies the directories to search for certain Tcl script files, including **init.tcl**. If specified, this becomes the value returned by the **info library** command.
TZ	Used to set the appropriate time zone for a host configured using the **dcecp host configure** command.
GDA Daemon Controls	
LDAP_AUTH_DN	Specifies the Distinguished Name (DN) in the LDAP Server that contains authentication information used to establish a connection to the LDAP Server. The authentication method used is LDAP_AUTH_SIMPLE. This environment variable is used by the GDA daemon and the **ldap_addcell** utility. In the case of the GDA daemon, the variable is optional for LDAP conduit, but required for **ldap_addcell**.
LDAP_AUTH_DN_PW	Specifies the password used during authentication when connecting to the LDAP Server. The password in stored in the Distinguished Name specified in LDAP_AUTH_DN. This environment variable is used by the GDA daemon and the **ldap_addcell** utility. In the case of the GDA daemon, the variable is optional for LDAP conduit, but required for **ldap_addcell**.
LDAP_SERVER	Used to indicate the IP address or TCP/IP host name of the LDAP Server in which DCE Cell information is registered. An example of this variable is: `LDAP_SERVER=`*`host`*`[:`*`port`*`]` where: *host* is the TCP/IP host name running the LDAP Server and *port* is the port on which the LDAP Server is listening. Alternatively, the host can be specified as an IP address. This environment variable is used by the GDA daemon and the **ldap_addcell** utility. In the case of the GDA daemon, it pertains to the LDAP conduit only.

RESOLVER_CONFIG	Indicates the dataset name that defines TCP/IP system parameters required by client programs.
	
	These are examples of this environment variable:
	
	`RESOLVER_CONFIG=TCPIP.DATA`
	`RESOLVER_CONFIG=`*`hlq`*`.TCPIP.DATA`
	
	Note: In the example above, *hlq* represents the high-level qualifier.
	
	This environment variable is used by the GDA daemon and pertains to the BIND conduit only.

Client and Server Controls	

Table 31 (Page 3 of 17). OS/390 DCE Environment Variables

Name	Description
BIND_PE_SITE	Determines if the Security server is located by using the namespace or by reading the **pe_site** file. The acceptable values are: **1** The Security server is looked up by reading the **pe_site** file. **0** The Security server is looked up through a regular CDS query (the default action).
RPC_DEFAULT_ENTRY	Designates the namespace entry that is used as a starting point in searching for binding information of compatible servers by import and lookup routines. There is no default value.
RPC_DEFAULT_ENTRY_SYNTAX	Specifies the syntax of the name provided in the environment variable RPC_DEFAULT_ENTRY. It is also used by the NSI routines which allow a default value for the name syntax parameter. Valid values are defined in the include file <dce/rpcbase.idl>: **0** use default **1** unknown (unsupported) **2** DECdns (unsupported) **3** DCE (the default value) **4** ISO OSI X.500 (unsupported) **5** DOD Internet Domain Name Server (unsupported) **6** UUID string (unsupported)
RPC_MAX_UDP_PACKET_SIZE	By default, the RPC runtime environment breaks large RPC calls into UDP packets with a maximum size of 8304 bytes, when the **ncadg_ip_udp** protocol is used. If larger packets are supported, the RPC_MAX_UDP_PACKET_SIZE environment variable can be set to the size desired. This environment variable can also be set to a lower value to prevent IP fragmentation of the UDP packets. This may be necessary if the packets are traversing either a network with extremely limited resources or a firewall that is misconfigured and dropping fragments.
RPC_RESTRICTED_PORTS	Restricts the dynamic assignment of server ports by the RPC runtime environment. The RPC runtime environment will only assign the ports that are specified for the designated protocol. This environment variable does not affect well-known endpoints. There is no default value.
RPC_UNSUPPORTED_NETADDRS	Specifies a colon-separated list of interface addresses that are not to be used by DCE.
RPC_UNSUPPORTED_NETIFS	Specifies a colon-separated list of interface names (returned by the SIOCGIFCONF ioctl() function) that are not to be used by DCE.
TRY_PE_SITE	Specifies the security replica search order. **0** The security replica search order consists of the CDS namespace followed by the **pe_site** file. **1** The security replica search order consists of the **pe_site** file followed by the CDS namespace. The default is **TRY_PE_SITE=0**.
_EUV_ECHO_STDIN	Used by dcecp, cdscp, rpccp, dtscp, acl_edit and rgy_edit. Intended for batch execution of these interactive utilities. Echoes input commands to the standard output file. Valid values are: **1** Enabled **0** Disabled (the default action)

Table 31 (Page 4 of 17). OS/390 DCE Environment Variables

Name	Description
_EUV_ENVAR_FILE	Used to override the envar file name. The default value is *$HOME*/**envar**, where *$HOME* is your home directory, defined in the POSIX segment of a RACF user ID.
_EUV_EXC_ABEND_DUMPS	A client and server control environment variable that specifies if a dump is taken during an ABEND exception. The valid values are: **0** No dump is taken for an exception. **1** A dump is only taken for an uncaught exception (if no CATCH or CATCH_ALL clause exists). **2** A dump is taken in all cases except for an explicit catch of an exception (if no CATCH clause exists). This is the default value.
_EUV_EXC_SW_DUMPS	A client and server control environment variable that specifies if a dump is taken during an exception raised by software. The valid values are: **0** No dump is taken for an exception. **1** A dump is only taken for an uncaught exception (if no CATCH or CATCH_ALL clause exists). This is the default value. **2** A dump is taken in all cases except for an explicit catch of an exception (if no CATCH clause exists).
_EUV_FTRACE	Activates function tracing within DCE or user code compiled with the TEST(NONE) compiler option. Valid values are: **1** Enabled **0** Disabled (the default action)
_EUV_FTRACE_DEPTH	Specifies the maximum function depth which will be traced. Valid values are numeric values between 0 and 32767. If _EUV_FTRACE_DEPTH is not defined or has a value of 0, there is no limit and all function calls will be traced.
_EUV_HOME	Used to override the home directory value specified in the POSIX segment of a RACF user ID. The default value is the home directory.
_EUV_RPC_COLLECT_LOCK_DATA	Determines whether data for obtaining and releasing RPC global locks and datagram packet pool locks is stored in global structures. When viewed in a dump, this data may assist in problem determination. Data collection is enabled if this environment variable is set to 1. Otherwise, it is disabled. Note that there is a significant performance impact when this data is collected.

Table 31 (Page 5 of 17). OS/390 DCE Environment Variables

Name	**Description**
_EUV_RPC_COMM_TIMEOUT	Used to override the communication timeout default value. The timeout value can be any integer from 0 (zero) to 10, which is the same range of integers accepted by the **rpc_mgmt_set_com_timeout** API. These integers represent a relative amount of time and apply to the use of all APIs rather than to the use of only the first API. The values are: **0** Attempts to communicate for 1 second. **1** Attempts to communicate for 2 seconds. **2** Attempts to communicate for 4 seconds. **3** Attempts to communicate for 8 seconds. **4** Attempts to communicate for 15 seconds. **5** Attempts to communicate for 30 seconds (default). **6** Attempts to communicate for 60 seconds. **7** Attempts to communicate for 120 seconds. **8** Attempts to communicate for 240 seconds. **9** Attempts to communicate for 480 seconds. **10** Attempts to communicate infinitely.
_EUV_SEC_KRB5CCNAME_FILE	Specifies the file that contains the KRB5CCNAME environment variable. The KRB5CCNAME environment variable contains the name of the Security credentials cache file. This environment variable enables a user to switch between multiple user identities. Primarily intended for non-shell environments (e.g. TSO, batch). The default value is *$HOME*/**krb5ccname**.
_EUV_USE_HOST_PROFILE	Specifies the CDS namespace search order. **0** The CDS namespace search order consists of the security group **(/.:/sec)** followed by the cell profile **(/.:/cell-profile)**. **1** The CDS namespace search order consists of the host profile **(/.:/hosts/<hostname>/profile)** followed by the security group **(/.:/sec)** followed by the cell profile **(/.:/cell-profile)**. This only applies when searching for a replica in the local cell. The default value is **_EUV_USE_HOST_PROFILE=0**.
Server Controls	
DCEAUDITFILTERON	Used by an Audit Client to indicate that filter information is to be applied to the audit event so that the audit event is recorded. Default is that filtering is not active.
DCEAUDITOFF	Indicates auditing is not active. Default is that auditing is active.
DCEAUDITTRAILSIZE	Defines the size of the audit trail size. Default size is 2 MB.
_EUV_LOAD_BALANCE	Provides the value of *aename* to the Runtime DLL for workload-balanced interfaces. This value is overridden by any value set using the **rpc_set_ae_name** API.
_EUV_RPC_ACL_FILE	Specifies the name of the ACL database file used by a server. No default exists.
| ***Security Server Controls***	
| SECD_DB2_CACHE_SIZE	Specifies the DB2® cache size in kilobytes and defaults to 4096 (yielding a cache size of 4MB). The DB2 cache is used to hold the results of DB2 registry queries.

Table 31 (Page 6 of 17). OS/390 DCE Environment Variables

Name	Description
SECD_DB2_SUBSYSTEM	Specifies the DB2 subsystem name and defaults to **DSN**. The subsystem name is limited to no more than 4 characters and must be the name assigned to the DB2 subsystem that contains the security registry database. This variable must be set for the **secd**, **sec_create_db**, **sec_export_db**, and **sec_import_db** commands when the security registry is stored in DB2.
SECD_DB2_THREADS	Specifies the number of DB2 threads and defaults to **4**. These threads are used to perform DB2 SQL requests when the security registry is stored in DB2. The DB2 DISPLAY THREAD(*) command can be used to monitor the activity of these threads.
SECD_CREDS_SIZE	Specifies the size of the security credentials dataspace in kilobytes and defaults to 20 480 (yielding a dataspace size of 20MB). The minimum value is 1024 and the maximum is 2 097 148.
SECD_KDC_THREADS	Specifies the number of KDC network threads and defaults to **5**. These threads are used to handle Kerberos requests from non-DCE clients.
SECD_KDC_USER_GROUP	Specifies the group to be used when a principal is created by the Kerberos **kadmin** command. The default group is **none**.
SECD_LOCAL_THREADS	Specifies the number of local server threads to be created and defaults to the number of RPC server threads. These threads are used to handle security requests from DCE clients on the same system as the security server.
SECD_THREADS	Specifies the number of RPC server threads to be created and defaults to **10**. These threads are used to handle security requests from remote DCE clients.
OS/390 Kernel Controls	
_EUV_DAEMONS_IN_AS	Specifies the list of daemons to run in their own address spaces instead of running within the DCEKERN address space. The eligible daemons are **secd**, **cdsd**, **dtstp**, **auditd**, **pwdmgmt**, and **gdad**. The default value for this variable is **secd cdsd**.
_EUV_HFS_MON	Specifies the time interval for monitoring utilization of the Hierarchical File System. If _EUV_HFS_MON is not defined, no monitoring will be performed.
_EUV_RACF_FACILITY_NAME	Specifies the name of the RACF facility that DCEKERN looks up to determine if a TSO user has DCEKERN start and stop permission. The default value is **DCEKERN.START.REQUEST**.

Table 31 (Page 7 of 17). OS/390 DCE Environment Variables

Name	Description
Message Controls	
_EUV_SVC_API_DUMPS	A message control environment variable that specifies if a dump is taken on parameter errors to public APIs. The valid values are: **1** Dumping is enabled (the default value). **0** Dumping is disabled.
_EUV_SVC_MSG_LEVEL	A message control environment variable that sets the minimum severity level of messages that are actually logged and displayed. The valid values are: **NONE** No messages are logged. **FATAL** Only fatal messages are logged. This value controls messages about non-recoverable errors. Usually, these messages are issued when a certain degree of permanent loss or damage occurs, such as the corruption of the database. **ERROR** Only error and fatal messages are logged. This value controls messages about unexpected events that are recoverable or that can be corrected by manual intervention. **USER** Only user, error, and fatal messages are logged. This value controls messages about errors detected in the use of a DCE Application Programming Interface (API). **WARNING** Only warning, user, error, and fatal messages are logged. This value controls messages about one of the following conditions: • An error occurred that was automatically corrected by the program or system. • A condition was detected which may be an error depending on whether the effects of the condition are acceptable. • A condition exists that if left uncorrected will eventually result in an error. **NOTICE** Only key informational messages are logged, along with all warning, error, user, and fatal messages. This value controls informational messages about major events such as the startup of a server. **VERBOSE** All messages are logged. This is the default action. This value controls informational messages about events which are important in monitoring DCE, such as the creation and deletion of RPC endpoints.

Table 31 (Page 8 of 17). OS/390 DCE Environment Variables

Name	Description
_EUV_SVC_MSG_LOGGING	A message control environment variable that controls where the messages are routed and displayed. Messages are routed to different destinations depending on whether the environment is shell, TSO, or batch. The valid values are: **NO_LOGGING** All messages are suppressed. **STDIO_LOGGING** VERBOSE, NOTICE, and WARNING messages are routed to the standard output file, either the screen or the standard output DD card, while error and other messages are routed to the standard error file, either the screen or the standard error DD card. This is the default value. **CONSOLE_LOGGING** All messages are routed to the standard output file, either the screen or the standard output DD card. USER, NOTICE, ERROR, and FATAL messages are also routed to the operator console. All messages are displayed in English. **Note:** DCEKERN and the processes running under it do not write any messages to STDERR. All messages from DCEKERN and its subprocesses are written to STDOUT.
_EUV_SVC_MSG_TIMESTAMP	Adds a timestamp to fatal, error, and user messages when the value STDIO_LOGGING is specified for the environment variable _EUV_SVC_MSG_LOGGING. The valid values are: **0** Disabled (the default value). **1** Enabled.
Single Sign-on Controls	
_EUV_AUTOLOG	Determines if single sign-on processing is ignored when an OS/390 user invokes a DCE application. The valid value is: **NO** Do not enable single sign-on processing. Any other value causes the _EUV_AUTOLOG environment to be ignored.
Debug Controls For the variables in this part of the table, there are debug levels 1 through 9 and 15. General usage of the levels is in pairs (1/2, 4/5, 7/8; levels 3, 6, and 9 are reserved). This does not include status codes, which may be at any level. The first level in the pair generally does not contain data. Data in the second level of the pair may include the contents of registers, binding handles, interfaces, and so forth, anything that can assist the person debugging the problem. Debug level 15 is a special debug level that is not maskable. Level 15 produces output regardless of the current debug level when the debug trace facility is enabled. For more information, see the *OS/390 DCE: Command Reference*.	

Table 31 (Page 9 of 17). OS/390 DCE Environment Variables

Name	**Description**
SVC_AUD_DBG	Sets the Audit daemon debug level. Valid values are 1 to 9. The following list is only a general guideline and may vary from one component to another. **0** Debug is off. **1** Entry and exit of components of DCE (such as runtime library, security). Also entry to callable APIs. **2** Data pertinent at the point of entry or exit of level 1 trace entries. **3** Reserved **4** Significant events, calls to functions outside of DCE, and calls which may go to a server on a different machine. **5** Data pertinent at the point of entry or exit of level 4 trace entries. **6** Reserved **7** Remaining debugs. **8** Data pertinent at the point of entry or exit of level 7 trace entries. **9** Reserved
SVC_CDS_DBG	Sets the CDS component debug level. Valid values are 1 to 9. The following list is only a general guideline and may vary from one component to another. **0** Debug is off. **1** Entry and exit of components of DCE (such as runtime library, security). Also entry to callable APIs. **2** Data pertinent at the point of entry or exit of level 1 trace entries. **3** Reserved **4** Significant events, calls to functions outside of DCE, and calls which may go to a server on a different machine. **5** Data pertinent at the point of entry or exit of level 4 trace entries. **6** Reserved **7** Remaining debugs. **8** Data pertinent at the point of entry or exit of level 7 trace entries. **9** Reserved

Table 31 (Page 10 of 17). OS/390 DCE Environment Variables

Name	Description
SVC_DHD_DBG	Sets the DCE daemon debug level. Valid values are 1 to 9. The following list is only a general guideline and may vary from one component to another. **0** Debug is off. **1** Entry and exit of components of DCE (such as runtime library, security). Also entry to callable APIs. **2** Data pertinent at the point of entry or exit of level 1 trace entries. **3** Reserved **4** Significant events, calls to functions outside of DCE, and calls which may go to a server on a different machine. **5** Data pertinent at the point of entry or exit of level 4 trace entries. **6** Reserved **7** Remaining debugs. **8** Data pertinent at the point of entry or exit of level 7 trace entries. **9** Reserved
SVC_DTS_DBG	Sets the DTS component debug level. Valid values are 1 to 9. The following list is only a general guideline and may vary from one component to another. **0** Debug is off. **1** Entry and exit of components of DCE (such as runtime library, security). Also entry to callable APIs. **2** Data pertinent at the point of entry or exit of level 1 trace entries. **3** Reserved **4** Significant events, calls to functions outside of DCE, and calls which may go to a server on a different machine. **5** Data pertinent at the point of entry or exit of level 4 trace entries. **6** Reserved **7** Remaining debugs. **8** Data pertinent at the point of entry or exit of level 7 trace entries. **9** Reserved

Table 31 (Page 11 of 17). OS/390 DCE Environment Variables

Name	Description
SVC_GSS_DBG	Sets the GSS API component debug level. Valid values are 1 to 9. The following list is only a general guideline and may vary from one component to another. **0** Debug is off. **1** Entry and exit of components of DCE (such as runtime library, security). Also entry to callable APIs. **2** Data pertinent at the point of entry or exit of level 1 trace entries. **3** Reserved **4** Significant events, calls to functions outside of DCE, and calls which may go to a server on a different machine. **5** Data pertinent at the point of entry or exit of level 4 trace entries. **6** Reserved **7** Remaining debugs. **8** Data pertinent at the point of entry or exit of level 7 trace entries. **9** Reserved
SVC_LIB_DBG	Sets the **acldb** debug level. Valid values are 1 to 9. The following list is only a general guideline and may vary from one component to another. **0** Debug is off. **1** Entry and exit of components of DCE (such as runtime library, security). Also entry to callable APIs. **2** Data pertinent at the point of entry or exit of level 1 trace entries. **3** Reserved **4** Significant events, calls to functions outside of DCE, and calls which may go to a server on a different machine. **5** Data pertinent at the point of entry or exit of level 4 trace entries. **6** Reserved **7** Remaining debugs. **8** Data pertinent at the point of entry or exit of level 7 trace entries. **9** Reserved

Table 31 (Page 12 of 17). OS/390 DCE Environment Variables

Name	Description
SVC_PLT_DBG	Sets the Platform component debug level. Valid values are 1 to 9. The following list is only a general guideline and may vary from one component to another. **0** Debug is off. **1** Entry and exit of components of DCE (such as runtime library, security). Also entry to callable APIs. **2** Data pertinent at the point of entry or exit of level 1 trace entries. **3** Reserved **4** Significant events, calls to functions outside of DCE, and calls which may go to a server on a different machine. **5** Data pertinent at the point of entry or exit of level 4 trace entries. **6** Reserved **7** Remaining debugs. **8** Data pertinent at the point of entry or exit of level 7 trace entries. **9** Reserved
SVC_RPC_DBG	Sets the RPC component debug level. Valid values are 1 to 9. The following list is only a general guideline and may vary from one component to another. **0** Debug is off. **1** Entry and exit of components of DCE (such as runtime library, security). Also entry to callable APIs. **2** Data pertinent at the point of entry or exit of level 1 trace entries. **3** Reserved **4** Significant events, calls to functions outside of DCE, and calls which may go to a server on a different machine. **5** Data pertinent at the point of entry or exit of level 4 trace entries. **6** Reserved **7** Remaining debugs. **8** Data pertinent at the point of entry or exit of level 7 trace entries. **9** Reserved

Table 31 (Page 13 of 17). OS/390 DCE Environment Variables

Name	Description
SVC_SEC_DBG	Sets the Security component debug level. Valid values are 1 to 9. The following list is only a general guideline and may vary from one component to another. **0** Debug is off. **1** Entry and exit of components of DCE (such as runtime library, security). Also entry to callable APIs. **2** Data pertinent at the point of entry or exit of level 1 trace entries. **3** Reserved **4** Significant events, calls to functions outside of DCE, and calls which may go to a server on a different machine. **5** Data pertinent at the point of entry or exit of level 4 trace entries. **6** Reserved **7** Remaining debugs. **8** Data pertinent at the point of entry or exit of level 7 trace entries. **9** Reserved
SVC_SED_DBG	Sets the Security Server daemon debug level. Valid values are 1 to 9. The following list is only a general guideline and may vary from one component to another. **0** Debug is off. **1** Entry and exit of components of DCE (such as runtime library, security). Also entry to callable APIs. **2** Data pertinent at the point of entry or exit of level 1 trace entries. **3** Reserved **4** Significant events, calls to functions outside of DCE, and calls which may go to a server on a different machine. **5** Data pertinent at the point of entry or exit of level 4 trace entries. **6** Reserved **7** Remaining debugs. **8** Data pertinent at the point of entry or exit of level 7 trace entries. **9** Reserved
_EUV_SVC_CEEDUMPS_LVL	This environment variable controls the amount of data captured in each CEEDUMP that is taken by a DCE probe. It does not affect dumps taken due to an exception. The CEEDUMP level can be specified as a full dump (this is the default), a partial dump, or no dump. Valid values are: • 0 - no dump • 5 - partial dump • 9 - full dump If dumps are suppressed (_EUV_SVC_CEEDUMPS_LVL=0), no dump is taken regardless of the settings of other environment variables, such as _EUV_SVC_API_DUMPS=1. In addition, the internal counter for the number of CEEDUMPs taken is not incremented when dumps are suppressed (see _EUV_SVC_CEEDUMPS_PER_PROCESS).

Table 31 (Page 14 of 17). OS/390 DCE Environment Variables

Name	Description
_EUV_SVC_CEEDUMPS_PER_PROCESS	A client and server control environment variable that specifies the maximum number of CEEDUMPS that will be generated per OS/390 DCE process. An OS/390 DCE process is either a DCEKERN daemon, a DCE control program (like **dcecp**), or a customer program link-edited with the DCE Runtime Library. This allows each DCE daemon to have its own limit. The valid values for _EUV_SVC_CEEDUMPS_PER_PROCESS are: **0** Turns off the limit checking. Unlimited dumping occurs. **1 to 2147483647** The number of CEEDUMPS you are willing to have. **A non-digit** Defaults to 0 (no limit). For example, if _EUV_SVC_CEEDUMPS_PER_PROCESS=OFF was specified, the result is unlimited dumping. **_EUV_SVC_CEEDUMPS_PER_PROCESS not specified** Defaults to 0 (unlimited). If a process tries to create another CEEDUMP and the limit is reached for that process, an information message is sent to the process **stdout** and to the operator console.
_EUV_SVC_DBG_MSG_CACHE	Enables caching of debug trace messages in memory for output at a later time when cache size and output destination are specified.
_EUV_SVC_DBG_MSG_LOGGING	Turns debug tracing on and off. Valid values are: **1** On **0** Off (the default value).
DCECONF Controls	
_EUV_CFG_AUDIT_FILE_NAME	Specifies the file name of the audit trail file. The file name must be 50 or fewer characters. It is truncated if you specify more than 50 characters. Only the auditd component uses this environment variable.
_EUV_CFG_AUDIT_FILE_PATH	Specifies the full path name of the directory where you want the audit trail file to reside. The path name must be 50 or fewer characters. It is truncated if you specify more than 50 characters. Only the auditd component uses this environment variable.
_EUV_CFG_AUDIT_FILE_WRAP	Specifies whether the audit daemon should wrap. Valid values are Y (yes) and N (no). N causes the audit daemon to open a new audit trail file when the current audit trail file reaches the maximum size. The current audit trail file is renamed and the new file is opened with the original name. The default is N. Only the auditd component uses this environment variable.
_EUV_CFG_AUDIT_OWN_EVENTS	Specifies whether the audit daemon should audit its own events. Valid values are Y (yes) and N (no). The default is N. Only the auditd component uses this environment variable.
_EUV_CFG_CDSD_MACHINE_ADDR	Specifies the cdsd machine Internet address on the configuration panel. There is no default.
_EUV_CFG_CDSD_MACHINE_NAME	Specifies the cdsd machine name on the configuration panel. There is no default.
_EUV_CFG_CELL_ID	Specifies the cell administrator ID on the configuration panel. The default value is **cell_admin**.
_EUV_CFG_CELL_NAME	Specifies the cell name on the configuration panel. There is no default.

Table 31 (Page 15 of 17). OS/390 DCE Environment Variables

Name	Description
_EUV_CFG_CELL_PW	Specifies the cell administrator's password. **Important Note:** Putting this password in a plain-text file can compromise the security of your cell. This environment variable must be set if you are configuring DCE with the **mkdce** operand instead of using interactive ISPF panels.
_EUV_CFG_CLEARINGHOUSE	Specifies the clearinghouse name for an additional **cdsd**. The default value is *hostname*_**ch**.
_EUV_CFG_DCE_MACHINE_NAME	Specifies the identifying name within the cell of the machine being configured. This can be the same as the TCP/IP host name, but it does not have to be. The default is the long TCP/IP host name (*hostname.domain*) of the local machine.
_EUV_CFG_DIRLIST	Specifies the list of directories to be replicated at configuration time by an additional CDS server (cds_second).
_EUV_CFG_GDAD_BIND	Specifies whether to configure the the bind conduit of the Global Directory Agent (GDA). Valid values are **Y** (yes) and **N** (no). The default is **Y**. If **Y** is specified, then RESOLVER_CONFIG must also be specified, either in the user's envar file, the **gdad** envar file, or on the GDAD Configuration Menu.
_EUV_CFG_GDAD_LDAP	Specifies whether to configure the the LDAP conduit of the Global Directory Agent (GDA). Valid values are **Y** (yes) and **N** (no). The default is **Y**. If **Y** is specified, then LDAP_SERVER, LDAP_AUTH_DN, and LDAP_AUTH_DN_PW must also be specified, either in the user's envar file, the **gdad** envar file, or on the GDAD Configuration Menu.
_EUV_CFG_INFORM_LEVEL	Specifies the level of informational data displayed during configuration. Valid values are: **0** Only progress messages are displayed. **1** Progress messages and commands are displayed (the default value). **2** Progress messages, commands and command output are displayed.
_EUV_CFG_KEYSEED	Specifies the keyseed for the initial security database master key used on the configuration panel. There is no default.
_EUV_CFG_LDAP_ADDCELL_DELETE	Specifies whether LDAP global cell registration should delete any existing data. Valid values are **Y** (yes) and **N** (no). The default is **N**.
_EUV_CFG_LOG_FILE	Specifies the history log file name. The default value is *$HOME*/**dceconf.log**.
_EUV_CFG_MAX_AUDIT_TRAIL	Specifies the maximum size of the audit trail file in bytes. The range is 1 to 429 467 294. Only the pwd component uses this environment variable.
_EUV_CFG_MAX_ID	Specifies the maximum value the Security server may automatically generate for a principal or group UNIX ID. The default value is **32 767** .
_EUV_CFG_MIN_PW_LTH	Specifies the minimum length of a principal's password. Specify 0 to indicate no minimum length. The maximum password length is 512 characters. Only the pwd component uses this environment variable.
_EUV_CFG_PW_ONLY_ALPHANUM	Specifies whether passwords should be limited to alphanumeric characters. Valid values are Y (yes) and N (no). The default is N. Only the pwd component uses this environment variable.

Table 31 (Page 16 of 17). OS/390 DCE Environment Variables

Name	Description
_EUV_CFG_PW_SPACE_OK	Specifies whether passwords can contain all spaces. Valid values are Y (yes) and N (no). The default is N. Only the pwd component uses this environment variable.
_EUV_CFG_REPLICANAME	Specifies the replica security server's name on the configuration panel. The default value is **cell_replica**.
_EUV_CFG_RGY_DB_TYPE	Specifies the type of registry database for the Security server. This is only used for component sec_srv. Valid values are HFS and RDB.
_EUV_CFG_RGY_INTERVAL	Specifies the checkpoint interval for the registry database. Only the sec_srv component uses this environment variable.
_EUV_CFG_SECD_DCE_MACHINE_NAME	Specifies the DCE host name of the Security server host. There is no default for this value. It is needed if you are configuring the initial CDS server on the OS/390 host and the initial Security server on another machine in the DCE cell.
_EUV_CFG_SECD_MACHINE_ADDR	Specifies the secd machine Internet address on the configuration panel. There is no default.
_EUV_CFG_SECD_MACHINE_NAME	Specifies the secd machine name on the configuration panel. There is no default.
_EUV_CFG_START_GID	Specifies the value at which the Security server starts assigning automatically-generated group UNIX IDs. The default value is **100** if a value is not specified on the configuration panel.
_EUV_CFG_START_OID	Specifies the value at which the Security server starts assigning automatically-generated organization UNIX IDs. The default value is **100**.
_EUV_CFG_START_UID	Specifies the value at which the Security server starts assigning automatically-generated principal UNIX IDs. The default value is **100** if a value is not specified on the configuration panel.
IDL Compiler Controls	
_EUV_IDL_COMPILER_NAME	Sets the C compiler load module. The default value is EDCDC120.
_EUV_IDL_WORKDA_UNIT	Specifies the work dataset unit for internal C compiler files. The default value is **VIO**.
Kerberos Controls **Note:** These variables are for use by the Kerberos service. The applications that use Kerberos can explicitly specify the values to be used, rather than allow Kerberos to use the default or environment variable. For example, DCE applications manage the value of the Kerberos controls, and do not support user-tailoring of the Kerberos environment variables.	
KRB5CCNAME	For DCE applications only, KRB5CCNAME is the default name for the credentials cache file. Specify it as "type:name." The supported types are FILE and MEMORY. When the user logs in to DCE, the **sec_login_set_context** API is issued and the Security service changes the value of KRB5CCNAME. This variable is automatically managed within the DCE library and should not be set by the user. In OS/390 DCE, KRB5CCNAME is not an actual environment variable. However, as KRB5CCNAME is an environment variable in other implementations of DCE, it is included here for reference purposes. See the _EUV_SEC_KRB5CCNAME_FILE environment variable on page 471 for information on how to control KRB5CCNAME within DCE applications.
KRB5RCACHEDIR	This is the default replay cache directory. The default value for this environment variable is **/tmp**.
KRB5RCACHENAME	This is the default replay cache name. If this is not specified, the Kerberos runtime generates a name.

Table 31 (Page 17 of 17). OS/390 DCE Environment Variables

Name	Description
KRB5RCACHETYPE	This is the default replay cache type. The default value for this environment variable is *dfl*.
KRB5_CONFIG	This environment variable consists of one or more configuration file names separated by colons. The default configuration file is **/krb5/krb5.conf**.
KRB5_KTNAME	This is the default key table name. If nothing is specified for this environment variable, the *default_keytab_name* configuration entry determines the file to be used. If the configuration entry is not used, the default key table name is **/krb5/v5srvtab**.

Format for Setting Environment Variables

Environment variables are set using the following format:

```
variable-name=value
```

For example,

```
BIND_PE_SITE=1
```

There should be no space between the *variable-name* or the *value* and the equal sign that separates them. If a definition is too long to fit in a single line, you can continue the declaration to the next line by appending a back slash (\) at the end of the current line. For example:

```
LONG_VARIABLE=this variable is far too long \
for one line.
```

How to Set Environment Variables

This section describes the three basic ways by which you can set environment variables. These are:

- By setting the environment variables in the **envar** file. Environment variables that are set in this file can be overridden using the next two methods.
- From the shell, using the export command.
- From batch or TSO, using the ENVAR runtime option.

Setting Variables in the Environment Variable File

You can set environment variables by editing the **default environment variable file**, also known as the **envar** file. This file contains the environment variable declarations, one declaration per line (unless the declaration string straddles multiple lines because of length).

This file must be created in the user's home directory with the name **envar**. (You can change the pathname of the default environment variable file by setting the value of the **_EUV_ENVAR_FILE** environment variable to a specific path name.)

Following is an example of the contents of the environment variable file:

```
BIND_PE_SITE=1
_EUV_ECHO_STDIN=1
RPC_DEFAULT_ENTRY=/.:/servers/server1
```

Figure 89. Example Environment Variable File

Environment variable declarations made through EXEC or CALL statements in TSO or batch (either by explicit declarations or pointing to an environment variable file) override the environment variable declarations in the default environment variable file while the program is executing. This is discussed in "Setting Environment Variables from Batch or TSO" on page 484.

Environment variables that are set through the **export** command in the shell environment override the default environment variable file while executing in the shell. This is discussed in "Setting Environment Variables from the Shell."

If you do not declare environment variables or point to an environment variable file in an EXEC or CALL statement in TSO or batch, or if you do not declare an environment variable using the export command in the shell, the program will use the default environment variable file in your home directory (with the name **envar**). If this file does not exist, the program assumes that there is no environment variable file.

Setting Environment Variables from the Shell

You can use the export command to set environment variables from the shell.

In OS/390 DCE, the export command sets a value for a variable and makes the variable available to the shell and to all processes forked by the shell. The common method of running this command is to include it in your **.profile** file which is run every time you enter the shell. For example, to set and export an environment variable, enter the following from the shell prompt or include it in your .profile file:

```
export BIND_PE_SITE=1
```

For more information on the export command or the .profile file, refer to *OS/390 UNIX System Services User's Guide*.

If you do not explicitly declare environment variables through the export command (from the command line or through the **.profile** file) in the shell, the default environment variable file (**envar** in your home directory) is used.

Setting Environment Variables from Batch or TSO

Environment variables can be set from batch or TSO by using the **ENVAR** runtime option. These are described in the following sections.

If you do not explicitly declare environment variables using any of the methods described in this section, the default environment variable file (**envar**) is used.

Setting Environment Variables from Batch:

In batch, the PARM statement will contain the ENVAR runtime options and the parameters that will be passed to the program. The "/" character separates the runtime options and the parameters. For example:

```
//PROG  EXEC PGM=PROG,PARM=('POSIX(ON),STACK(12000),ENVAR(''var1=value
//             1'' ''var2=value2'')/pgm-parms')
```

Note: The ENVAR declaration is enclosed in two single quotation marks, not double quotation marks.

The length of the passed parameters on the PARM statement is limited to 100 characters and if the ENVAR and other PARM declarations exceed this limit, an alternative way must be used. The alternative is to point to a file (other than the default environment variable file) that contains all the variable declarations using the format:

```
...ENVAR(''_EUV_ENVAR_FILE=filename'')
```

where *filename* is the name of the HFS file that contains all the variable declarations, one declaration per line. If this is a relative HFS pathname, the file must exist in the user's home directory.

You can also refer to the environment variable file using the following format:

```
...ENVAR(''_EUV_ENVAR_FILE=//DD:filename'')...
```

where *filename* is the symbolic name of a DD statement that specifies the file containing the environment variables.

Setting Environment Variables from TSO: Environment variables can be set when CALLing a program through the ENVAR runtime option using the following format:

```
CALL MYPROG 'POSIX(ON),STACK(12000),ENVAR(''var1=value1'' ''var2=value2'')/pgm-parms'
```

As in batch, ENVAR can also be made to point to a file (other than the default environment variable file). For example:

```
CALL MYPROG 'POSIX(ON),ENVAR("_EUV_ENVAR_FILE=file1")/pgm-parms'
```

Again, the environment variable file is an HFS file.

OS/390 DCE Daemon Environment Variable Files

DCEKERN and each OS/390 DCE daemon has its own home directory. Each home directory contains the **environment variable file** (**envar**), where the environment variables are set for each of the daemons.

The **envar** files of the OS/390 DCE daemons are in:

- /opt/dcelocal/home/auditd/envar
- /opt/dcelocal/home/cdsadv/envar
- /opt/dcelocal/home/cdsclerk/envar
- /opt/dcelocal/home/dced/envar
- /opt/dcelocal/home/dcekern/envar
- /opt/dcelocal/home/dtsd/envar
- /opt/dcelocal/home/dts_null_provider/envar
- /opt/dcelocal/home/pwdmgmt/envar
- /opt/dcelocal/home/secd/envar

You can customize the values of the environment variables in these files to suit your operational needs.

Messaging Subsystem Environment Variables

OS/390 DCE provides a messaging facility which displays messages from DCE services or DCE applications. You can control the display of these messages based on the severity level using the **_EUV_SVC_MSG_LEVEL** environment variable. You can also control where the messages are displayed using the **_EUV_SVC_MSG_LOGGING** environment variable.

Appendix B. The Code Set Registry

The code set registry is a file on each host machine that maps the code set names that the host operating system supports to the unique identifiers for those code sets that were assigned by the Open Software Foundation (OSF).

Character Sets and Code Sets

A character set is a group of characters, such as the Latin-1 alphabet or Japanese Kanji. A code set is a mapping of the members of a character set to specific numeric code values. Examples of code sets include ASCII, IBM-939 (Japanese Kanji), and ISO 8859-1 (Latin 1).

Note: The OS/390 operating system may call a code set by one name, and another operating system may refer to the same code set by another name. For example, OS/390 uses "ISO8859-1" for the ISO encoding of the Latin-1 alphabet, while another operating system might use "Latin-1."

Code Sets and DCE

Until this release of OS/390 DCE, the only way that RPC applications were able to send character data over networks was by using characters from the POSIX Portable Character Set. These characters are converted from the EBCDIC code page used by the application to ISO 8859-1 (ASCII) before being sent over the network.

With the new OS/390 DCE RPC Internationalization support, RPC applications are now also able to send character data that is not restricted to the POSIX Portable Character Set. The RPC applications can use any code page that is supported by the host system. Evaluation checks are performed on the data so that a massive data loss does not occur during this transfer. For information on how to use the Internationalization support, see *OS/390 DCE: Application Development Guide: Core Components*.

What is the DCE Code Set Registry?

The DCE code set registry provides a mechanism for uniquely identifying code sets and the character sets they encode across multiple heterogeneous operating systems. The code set registry is a file on each host machine that maps the code set names supported by that machine to the OSF-assigned unique identifiers for those code sets. Assigning a unique identifier to a code set provides internationalized DCE RPC clients and servers with a common representation to use when referring to a given code set.

The DCE Code Set Registry in OS/390

OS/390 DCE provides a code set registry in the form of a binary file that is shipped with the product code. The fully-qualified name for this file is **/usr/lib/nls/csr/code_set_registry.db**. The information in the binary file is shown in Table 32 on page 488. Note that OS/390 DCE does not support every possible code set. Those that are not supported are shown as "none" in the column marked "Code Set Name" in the table.

Table 32 (Page 1 of 10). The Code Set Registry

Description	Code Set Value Registered by OSF	OS/390 Support?	Code Set Name	Character Sets Supported by This Code Set (hex values set by OSF)	Maximum Bytes for This Code Set
International or national standard code sets/encoding methods					
ISO 8859-1:1987; Latin Alphabet No. 1	0x00010001	Y	IBM-819	0x0011	1
ISO 8859-2:1987; Latin Alphabet No. 2	0x00010002	Y	IBM-912	0x0012	1
ISO 8859-3:1988; Latin Alphabet No. 3	0x00010003	N	none	0x0013	1
ISO 8859-4:1988; Latin Alphabet No. 4	0x00010004	Y	IBM-914	0x0014	1
ISO/IEC 8859-5:1988; Latin-Cyrillic Alphabet	0x00010005	Y	IBM-915	0x0015	1
ISO 8859-6:1987; Latin-Arabic Alphabet	0x00010006	Y	IBM-1089	0x0016	1
ISO 8859-7:1987; Latin-Greek Alphabet	0x00010007	Y	IBM-813	0x0017	1
ISO 8859-8:1988; Latin-Hebrew Alphabet	0x00010008	Y	IBM-916	0x0018	1
ISO/IEC 8859-9:1989; Latin Alphabet No. 5	0x00010009	Y	IBM-920	0x0019	1
ISO/IEC 8859-10:1992; Latin Alphabet No. 6	0x0001000a	N	none	0x001a	1
ISO 646:1991 IRV (International Reference Version)	0x00010020	N	none	0x0001	1
ISO/IEC 10646-1:1993; UCS-2, Level 1	0x00010100	Y	UCS-2	0x1000	2
ISO/IEC 10646-1:1993; UCS-2, Level 2	0x00010101	N	none	0x1000	2
ISO/IEC 10646-1:1993; UCS-2, Level 3	0x00010102	N	none	0x1000	2
ISO/IEC 10646-1:1993; UCS-4, Level 1	0x00010104	N	none	0x1000	4
ISO/IEC 10646-1:1993; UCS-4, Level 2	0x00010105	N	none	0x1000	4
ISO/IEC 10646-1:1993; UCS-4, Level 3	0x00010106	N	none	0x1000	4
ISO/IEC 10646-1:1993; UTF-1, UCS Transformation Format 1	0x00010108	N	none	0x1000	5
JIS X0201:1976; Japanese phonetic characters	0x00030001	N	none	0x0080	1
JIS X0208:1978 Japanese Kanji Graphic Characters	0x00030004	N	none	0x0081	2

Table 32 (Page 2 of 10). The Code Set Registry

Description	**Code Set Value Registered by OSF**	**OS/390 Support?**	**Code Set Name**	**Character Sets Supported by This Code Set (hex values set by OSF)**	**Maximum Bytes for This Code Set**
JIS X0208:1983 Japanese Kanji Graphic Characters	0x00030005	N	none	0x0081	2
JIS X0208:1990 Japanese Kanji Graphic Characters	0x00030006	N	none	0x0081	2
JIS X0212:1990; Supplementary Japanese Kanji Graphic Chars	0x0003000a	N	none	0x0082	2
JIS eucJP:1993; Japanese EUC	0x00030010	Y	eucJP	0x0011, 0x0080, 0x0081, 0x0082	3
KS C5601:1987; Korean Hangul and Hanja Graphic Characters	0x00040001	N	none	0x0100	2
KS C5657:1991; Supplementary Korean Graphic Characters	0x00040002	N	none	0x0101	2
KS eucKR:1991; Korean EUC	0x0004000a	Y	eucKR	0x0011, 0x0100, 0x0101	2
CNS 11643:1986; Taiwanese Hanzi Graphic Characters	0x00050001	N	none	0x0180	2
CNS 11643:1992; Taiwanese Extended Hanzi Graphic Chars	0x00050002	N	none	0x0181	4
CNS eucTW:1991; Taiwanese EUC	0x0005000a	N	none	0x0001, 0x0180	4
CNS eucTW:1993; Taiwanese EUC	0x00050010	Y	eucTW-1993	0x0001, 0x0181	4
TIS 620-2529, Thai characters	0x000b0001	Y	TIS-620	0x0200	1
TTB CCDC:1984; Chinese Code for Data Communications	0x000d0001	N	none	0x0180	2
Industry consortium code sets/encoding methods					
OSF Japanese UJIS	0x05000010	N	none	0x0001, 0x0080, 0x0081	2
OSF Japanese SJIS-1	0x05000011	N	none	0x0001, 0x0080, 0x0081	2
OSF Japanese SJIS-2	0x05000012	N	none	0x0001, 0x0080, 0x0081	2
X/Open FSS-UTF; File System Safe UCS Trans. Format for ISO 10646-1	0x05010001	N	none	0x1000	6
JVC_eucJP	0x05020001	N	none	0x0001, 0x0080, 0x0081, 0x0082	3
JVC_SJIS	0x05020002	N	none	0x0001, 0x0080, 0x0081	2
Commercial company code sets/encoding methods					

Table 32 (Page 3 of 10). The Code Set Registry

Description	Code Set Value Registered by OSF	OS/390 Support?	Code Set Name	Character Sets Supported by This Code Set (hex values set by OSF)	Maximum Bytes for This Code Set
DEC Kanji	0x10000001	N	none	0x0011, 0x0080, 0x0081	2
Super DEC Kanji	0x10000002	N	none	0x0011, 0x0080, 0x0081, 0x0082	3
DEC Shift JIS	0x10000003	N	none	0x0011, 0x0080, 0x0081	2
HP roman8; English and Western European languages	0x10010001	N	none	0x0011	1
HP kana8; Japanese katakana (incl JIS X0201:1976)	0x10010002	N	none	0x0080	1
HP arabic8; Arabic characters	0x10010003	N	none	0x0016	1
HP greek8; Greek characters	0x10010004	N	none	0x0017	1
HP hebrew8; Hebrew characters	0x10010005	N	none	0x0018	1
HP turkish8; Turkish characters	0x10010006	N	none	0x0013, 0x0019	1
HP15CN; encoding method for Simplified Chinese	0x10010007	N	none	0x0001, 0x0300	2
HP big5; encoding method for Traditional Chinese	0x10010008	N	none	0x0001, 0x0180	2
HP japanese15 (sjis); Shift-JIS for mainframe (incl JIS X0208:1990)	0x10010009	N	none	0x0001, 0x0080, 0x0081	2
HP sjishi; Shift-JIS for HP user (incl JIS X0208:1990)	0x1001000a	N	none	0x0001, 0x0080, 0x0081	2
HP sjispc; Shift-JIS for PC (incl JIS X0208:1990)	0x1001000b	N	none	0x0001, 0x0080, 0x0081	2
HP ujis; EUC (incl JIS X0208:1990)	0x1001000c	N	none	0x0001, 0x0080, 0x0081	2
IBM-037 (CCSID 00037); CECP for USA, Canada, NL, Ptgl, Brazil, Australia, NZ	0x10020025	Y	IBM-037	0x0011	1
IBM-273 (CCSID 00273); CECP for Austria, Germany	0x10020111	Y	IBM-273	0x0011	1
IBM-277 (CCSID 00277); CECP for Denmark, Norway	0x10020115	Y	IBM-277	0x0011	1
IBM-278 (CCSID 00278); CECP for Finland, Sweden	0x10020116	Y	IBM-278	0x0011	1
IBM-280 (CCSID 00280); CECP for Italy	0x10020118	Y	IBM-280	0x0011	1
IBM-282 (CCSID 00282); CECP for Portugal	0x1002011a	Y	IBM-282	0x0011	1

Table 32 (Page 4 of 10). The Code Set Registry

Description	Code Set Value Registered by OSF	OS/390 Support?	Code Set Name	Character Sets Supported by This Code Set (hex values set by OSF)	Maximum Bytes for This Code Set
IBM-284 (CCSID 00284); CECP for Spain, Latin America (Spanish)	0x1002011c	Y	IBM-284	0x0011	1
IBM-285 (CCSID 00285); CECP for United Kingdom	0x1002011d	Y	IBM-285	0x0011	1
IBM-290 (CCSID 00290); Japanese Katakana Host Ext SBCS	0x10020122	Y	IBM-290	0x0080	1
IBM-297 (CCSID 00297); CECP for France	0x10020129	Y	IBM-297	0x0011	1
IBM-300 (CCSID 00300); Japanese Host DBCS incl 4370 UD	0x1002012c	Y	IBM-300	0x0081	2
IBM-301 (CCSID 00301); Japanese PC Data DBCS incl 1880 UDC	0x1002012d	Y	IBM-301	0x0081	2
IBM-420 (CCSID 00420); Arabic (presentation shapes)	0x100201a4	Y	IBM-420	0x0016	1
IBM-424 (CCSID 00424); Hebrew	0x100201a8	Y	IBM-424	0x0018	1
IBM-437 (CCSID 00437); PC USA	0x100201b5	N	none	0x0011	1
IBM-500 (CCSID 00500); CECP for Belgium, Switzerland	0x100201f4	Y	IBM-500	0x0011	1
IBM-833 (CCSID 00833); Korean Host Extended SBCS	0x10020341	Y	IBM-833	0x0001	1
IBM-834 (CCSID 00834); Korean Host DBCS incl 1227 UDC	0x10020342	Y	IBM-834	0x0100	2
IBM-835 (CCSID 00835); T-Ch Host DBCS incl 6204 UDC	0x10020343	Y	IBM-835	0x0180	2
IBM-836 (CCSID 00836); S-Ch Host Extended SBCS	0x10020344	Y	IBM-836	0x0001	1
IBM-837 (CCSID 00837); S-Ch Host DBCS incl 1880 UDC	0x10020345	Y	IBM-837	0x0300	2
IBM-838 (CCSID 00838); Thai Host Extended SBCS	0x10020346	Y	IBM-838	0x0200	1
IBM-839 (CCSID 00839); Thai Host DBCS incl 374 UDC	0x10020347	N	none	0x0200	2
IBM-850 (CCSID 00850); Multilingual IBM PC Data-MLP 222	0x10020352	Y	IBM-850	0x0011	1

Table 32 (Page 5 of 10). The Code Set Registry

Description	**Code Set Value Registered by OSF**	**OS/390 Support?**	**Code Set Name**	**Character Sets Supported by This Code Set (hex values set by OSF)**	**Maximum Bytes for This Code Set**
IBM-852 (CCSID 00852); Multilingual Latin-2	0x10020354	Y	IBM-852	0x0012	1
IBM-855 (CCSID 00855); Cyrillic PC Data	0x10020357	Y	IBM-855	0x0015	1
IBM-856 (CCSID 00856); Hebrew PC Data (extensions)	0x10020358	Y	IBM-856	0x0018	1
IBM-857 (CCSID 00857); Turkish Latin-5 PC Data	0x10020359	N	none	0x0019	1
IBM-861 (CCSID 00861); PC Data Iceland	0x1002035d	Y	IBM-861	0x0011	1
IBM-862 (CCSID 00862); PC Data Hebrew	0x1002035e	Y	IBM-862	0x0018	1
IBM-863 (CCSID 00863); PC Data Canadian French	0x1002035f	N	none	0x0011	1
IBM-864 (CCSID 00864); Arabic PC Data	0x10020360	Y	IBM-864	0x0016	1
IBM-866 (CCSID 00866); PC Data Cyrillic 2	0x10020362	Y	IBM-866	0x0015	1
IBM-868 (CCSID 00868); Urdu PC Data	0x10020364	N	none	0x0016	1
IBM-869 (CCSID 00869); Greek PC Data	0x10020365	Y	IBM-869	0x0017	1
IBM-870 (CCSID 00870); Multilingual Latin-2 EBCDIC	0x10020366	Y	IBM-870	0x0012	1
IBM-871 (CCSID 00871); CECP for Iceland	0x10020367	Y	IBM-871	0x0011	1
IBM-874 (CCSID 00874); Thai PC Display Extended SBCS	0x1002036a	Y	IBM-874	0x0200	1
IBM-875 (CCSID 00875); Greek	0x1002036b	Y	IBM-875	0x0017	1
IBM-880 (CCSID 00880); Multilingual Cyrillic	0x10020370	Y	IBM-880	0x0015	1
IBM-891 (CCSID 00891); Korean PC Data SBCS	0x1002037b	Y	none	0x0001	1
IBM-896 (CCSID 00896); Japanese Katakana; JISX0201:1976	0x10020380	N	none	0x0080	1
IBM-897 (CCSID 00897); PC Data Japanese SBCS (use with 301)	0x10020381	N	none	0x0080	1
IBM-903 (CCSID 00903); PC Data Simplified Chinese SBCS	0x10020387	N	none	0x0001	1

Table 32 (Page 6 of 10). The Code Set Registry

Description	Code Set Value Registered by OSF	OS/390 Support?	Code Set Name	Character Sets Supported by This Code Set (hex values set by OSF)	Maximum Bytes for This Code Set
IBM-904 (CCSID 00904); PC Data Traditional Chinese SBCS	0x10020388	Y	IBM-904	0x0001	1
IBM-918 (CCSID 00918); Urdu	0x10020396	N	none	0x0016	1
IBM-921 (CCSID 00921); Baltic 8-Bit	0x10020399	Y	IBM-921	0x001a	1
IBM-922 (CCSID 00922); Estonia 8-Bit	0x1002039a	Y	IBM-922	0x001a	1
IBM-926 (CCSID 00926); Korean PC Data DBCS incl 1880 UDC	0x1002039e	N	none	0x0100	2
IBM-927 (CCSID 00927); T-Ch PC Data DBCS incl 6204 UDC	0x1002039f	Y	IBM-927	0x0180	2
IBM-928 (CCSID 00928); S-Ch PC Data DBCS incl 1880 UDC	0x100203a0	Y	IBM-928	0x0300	2
IBM-929 (CCSID 00929); Thai PC Data DBCS incl 374 UDC	0x100203a1	N	none	0x0200	2
IBM-930 (CCSID 00930); Kat-Kanji Host MBCS Ext-SBCS	0x100203a2	Y	IBM-930	0x0080, 0x0081	2
IBM-897 and -301 (CCSID 00932); Japanese PC Data Mixed	0x100203a4	Y	IBM-932	0x0080, 0x0081	2
IBM-833 and -834 (CCSID 00933); Korean Host Extended SBCS	0x100203a5	Y	IBM-933	0x0001, 0x0100	2
IBM-891 and -926 (CCSID 00934); Korean PC Data Mixed incl 1880 UDC	0x100203a6	N	none	0x0001, 0x0100	2
IBM-836 and -837 (CCSID 00935); S-Ch Host Mixed incl 1880 UDC	0x100203a7	Y	IBM-935	0x0001, 0x0300	2
IBM-936 (CCSID 00936); PC Data S-Ch MBCS	0x100203a8	Y	IBM-936	0x0001, 0x0300	2
IBM-037 and -835 (CCSID 00937); T-Ch Host Mixed incl 6204 UDC	0x100203a9	Y	IBM-937	0x0001, 0x0180	2
IBM-938 (CCSID 00938); PC Data T-Ch MBCS	0x100203aa	Y	IBM-938	0x0001, 0x0180	2
IBM-939 (CCSID 00939); Latin-Kanji Host MBCS	0x100203ab	Y	IBM-939	0x0080, 0x0081	2
IBM-941 (CCSID 00941); Japanese PC DBCS for Open	0x100203ad	N	none	0x0081	2

Table 32 (Page 7 of 10). The Code Set Registry

Description	Code Set Value Registered by OSF	OS/390 Support?	Code Set Name	Character Sets Supported by This Code Set (hex values set by OSF)	Maximum Bytes for This Code Set
IBM-1041 and -301 (CCSID 00942); Japanese PC Data Mixed	0x100203ae	Y	IBM-942	0x0080, 0x0081	2
IBM-943 (CCSID 00943); Japanese PC MBCS for Open	0x100203af	N	none	0x0080, 0x0081	2
IBM-1042 and -928 (CCSID 00946); S-Ch PC Data Mixed incl 1880 UDC	0x100203b2	Y	IBM-946	0x0001, 0x0300	2
IBM-947 (CCSID 00947); T-Ch PC Data DBCS incl 6204 UDC	0x100203b3	Y	IBM-947	0x0180	2
IBM-1043 and -927 (CCSID 00948); T-Ch PC Data Mixed incl 6204 UDC	0x100203b4	Y	IBM-948	0x0001, 0x0180	2
IBM-1088 and -951 (CCSID 00949); IBM KS PC Data Mixed	0x100203b5	Y	IBM-949	0x0001, 0x0100	2
IBM-1114 and -947 (CCSID 00950); T-Ch PC Data Mixed incl 6204 UDC	0x100203b6	Y	IBM-950	0x0001, 0x0180	2
IBM-951 (CCSID 00951); IBM KS PC Data DBCS incl 1880 UDC	0x100203b7	Y	IBM-951	0x0100	2
IBM-955 (CCSID 00955); Japan Kanji; JISX0208:1978	0x100203bb	N	none	0x0081	2
IBM-964 (CCSID 00964); T-Chinese EUC CNS1163 plane 1,2	0x100203c4	Y	IBM-eucTW	0x0001, 0x0180	4
IBM-970 (CCSID 00970); Korean EUC	0x100203ca	Y	IBM-eucKR	0x0011, 0x0100, 0x0101	2
IBM-1006 (CCSID 01006); Urdu 8-bit	0x100203ee	N	none	0x0016	1
IBM-1025 (CCSID 01025); Cyrillic Multilingual	0x10020401	Y	IBM-1025	0x0015	1
IBM-1026 (CCSID 01026); Turkish Latin-5	0x10020402	Y	IBM-1026	0x0019	1
IBM-1027 (CCSID 01027); Japanese Latin Host Ext SBCS	0x10020403	Y	IBM-1027	0x0080	1
IBM-1040 (CCSID 01040); Korean PC Data Extended SBCS	0x10020410	N	none	0x0001	1
IBM-1041 (CCSID 01041); Japanese PC Data Extended SBCS	0x10020411	N	none	0x0080	1

Table 32 (Page 8 of 10). The Code Set Registry

Description	Code Set Value Registered by OSF	OS/390 Support?	Code Set Name	Character Sets Supported by This Code Set (hex values set by OSF)	Maximum Bytes for This Code Set
IBM-1043 (CCSID 01043); T-Ch PC Data Extended SBCS	0x10020413	N	none	0x0001	1
IBM-1046 (CCSID 01046); Arabic PC Data	0x10020416	Y	IBM-1046	0x0016	1
IBM-1047 (CCSID 01047); Latin-1 Open System	0x10020417	Y	IBM-1047	0x0011	1
IBM-1088 (CCSID 01088); IBM KS Code PC Data SBCS	0x10020440	Y	IBM-1088	0x0001	1
IBM-1097 (CCSID 01097); Farsi	0x10020449	N	none	0x0016	1
IBM-1098 (CCSID 01098); Farsi PC Data	0x1002044a	N	none	0x0016	1
IBM-1112 (CCSID 01112); Baltic Multilingual	0x10020458	Y	IBM-1112	0x001a	1
IBM-1114 (CCSID 01114); T-Ch PC Data SBCS (IBM BIG-5)	0x1002045a	N	none	0x0001	1
IBM-1115 (CCSID 01115); S-Ch PC Data SBCS (IBM GB)	0x1002045b	Y	IBM-1115	0x0001	1
IBM-1122 (CCSID 01122); Estonia	0x10020462	Y	IBM-1122	0x001a	1
IBM-1250 (CCSID 01250); MS Windows Latin-2	0x100204e2	Y	IBM-1250	0x0012	1
IBM-1251 (CCSID 01251); MS Windows Cyrillic	0x100204e3	Y	IBM-1251	0x0015	1
IBM-1252 (CCSID 01252); MS Windows Latin-1	0x100204e4	Y	IBM-1252	0x0011	1
IBM-1253 (CCSID 01253); MS Windows Greek	0x100204e5	Y	IBM-1253	0x0017	1
IBM-1254 (CCSID 01254); MS Windows Turkey	0x100204e6	N	none	0x0019	1
IBM-1255 (CCSID 01255); MS Windows Hebrew	0x100204e7	Y	IBM-1255	0x0018	1
IBM-1256 (CCSID 01256); MS Windows Arabic	0x100204e8	Y	IBM-1256	0x0016	1
IBM-1257 (CCSID 01257); MS Windows Baltic	0x100204e9	N	none	0x001a	1
IBM-1380 (CCSID 01380); S-Ch PC Data DBCS incl 1880 UDC	0x10020564	Y	IBM-1380	0x0300	2
IBM-1115 and -1380 (CCSID 01381); S-Ch PC Data Mixed incl 1880 UDC	0x10020565	Y	IBM-1381	0x0001, 0x0300	2

Table 32 (Page 9 of 10). The Code Set Registry

Description	Code Set Value Registered by OSF	OS/390 Support?	Code Set Name	Character Sets Supported by This Code Set (hex values set by OSF)	Maximum Bytes for This Code Set
IBM-1383 (CCSID 01383); S-Ch EUC GB 2312-80 set (1382)	0x10020567	Y	IBM-eucCN	0x0001, 0x0300	3
IBM-300 (CCSID 04396); Japanese Host DBCS incl 1880 UDC	0x1002112c	Y	IBM-4396	0x0081	2
IBM-850 (CCSID 04946); Multilingual IBM PC Data-190	0x10021352	Y	IBM-4946	0x0011	1
IBM-852 (CCSID 04948); Latin-2 Personal Computer	0x10021354	N	none	0x0012	1
IBM-855 (CCSID 04951); Cyrillic Personal Computer	0x10021357	N	none	0x0015	1
IBM-856 (CCSID 04952); Hebrew PC Data	0x10021358	N	none	0x0018	1
IBM-857 (CCSID 04953); Turkish Latin-5 PC Data	0x10021359	N	none	0x0019	1
IBM-864 (CCSID 04960); Arabic PC Data (all shapes)	0x10021360	N	none	0x0016	1
IBM-868 (CCSID 04964); PC Data for Urdu	0x10021364	N	none	0x0016	1
IBM-869 (CCSID 04965); Greek PC Data	0x10021365	N	none	0x0017	1
IBM-290 and -300 (CCSID 05026); Japanese Katakana-Kanji Host Mixed	0x100213a2	Y	IBM-5026	0x0080, 0x0081	2
IBM-836 & IBM-837 (CCSID 05031); S-Ch Host MBCS	0x100213a7	Y	IBM-5031	0x0001, 0x0300	2
IBM-1027 and -300 (CCSID 05035); Japanese Latin-Kanji Host Mixed	0x100213ab	Y	IBM-5035	0x0080, 0x0081	2
IBM-5048 (CCSID 05048); Japanese Kanji; JISX0208:1990(&1983)	0x100213b8	N	none	0x0081	2
IBM-5049 (CCSID 05049); Japanese Kanji; JISX0212:1990	0x100213b9	N	none	0x0082	2
IBM-5067 (CCSID 05067); Korean Hangul and Hanja; KSC5601:1987	0x100213cb	N	none	0x0100	2
IBM-420 (CCSID 08612); Arabic (base shapes only)	0x100221a4	N	none	0x0016	1
IBM-833 (CCSID 09025); Korean Host SBCS	0x10022341	N	none	0x0001	1

Table 32 (Page 10 of 10). The Code Set Registry

Description	Code Set Value Registered by OSF	OS/390 Support?	Code Set Name	Character Sets Supported by This Code Set (hex values set by OSF)	Maximum Bytes for This Code Set
IBM-834 (CCSID 09026); Korean Host DBCS incl 1880 UDC	0x10022342	N	none	0x0100	2
IBM-838 (CCSID 09030); Thai Host Extended SBCS	0x10022346	N	none	0x0200	1
IBM-864 (CCSID 09056); Arabic PC Data (unshaped)	0x10022360	N	none	0x0016	1
IBM-874 (CCSID 09066); Thai PC Display Extended SBCS	0x1002236a	N	none	0x0200	1
IBM-833 and -834 (CCSID 09125); Korean Host Mixed incl 1880 UDC	0x100223a5	N	none	0x0001, 0x0100	2
IBM-850 (CCSID 25426); Multilingual IBM PC Display-MLP	0x10026352	N	none	0x0011	1
IBM-856 (CCSID 25432); Hebrew PC Display (extensions)	0x10026358	N	none	0x0018	1
IBM-1042 (CCSID 25618); S-Ch PC Display Ext SBCS	0x10026412	N	none	0x0001	1
IBM-037 (CCSID 28709); T-Ch Host Extended SBCS	0x10027025	Y	IBM-28709	0x0001	1
IBM-856 (CCSID 33624); Hebrew PC Display	0x10028358	N	none	0x0018	1
IBM33722 (CCSID 33722); Japanese EUC JISx201,208,212	0x100283ba	Y	IBM-eucJP	0x0080, 0x0081, 0x0082	3
HTCsjis : Hitachi SJIS 90-1	0x10030001	N	none	0x0001, 0x0080, 0x0081	2
HTCujis : Hitachi eucJP 90-1	0x10030002	N	none	0x0001, 0x0080, 0x0081	2

Appendix C. The DCE Cell Namespace

This appendix describes the names that CDS and the DCE Security Service use within the DCE cell namespace. Most of these namespace entries are created during initial DCE configuration of the cell.

In the tables that follow, the **CDS Class** field is either used internally by the **CDS_clearinghouse** entry or by the CDS Browser.

Note: CDS Browser is not available in OS/390 DCE.

The **Well known** field specifies whether the last component of a name is a required name. The **Initial Configuration ACL** field specifies the ACL created during the initial configuration of the DCE cell. The **Created by** field specifies how this entry is created.

The *hostname*, *lclhostname*, *cellname*, and *creator* entries are defined as follows:

hostname	This is a cell-relative 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**.
lclhostname	This is the single component host name. This name is always the least significant component of the host name. The *lclhostname* for the examples given previously are **abc** and **apollo**.
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.

The Cell Directory Service Namespace

Figure 90 and Figure 91 on page 500 depict the CDS namespace within the DCE cell namespace. Following the figures is a description of each entry.

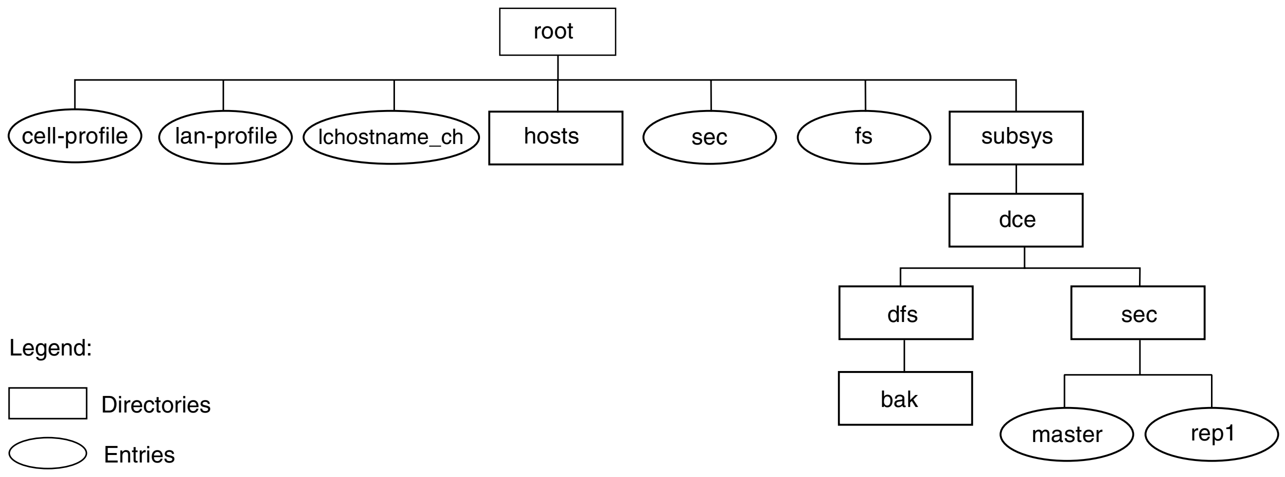


Figure 90. The Top of the CDS Namespace and the SUBSYS Directory

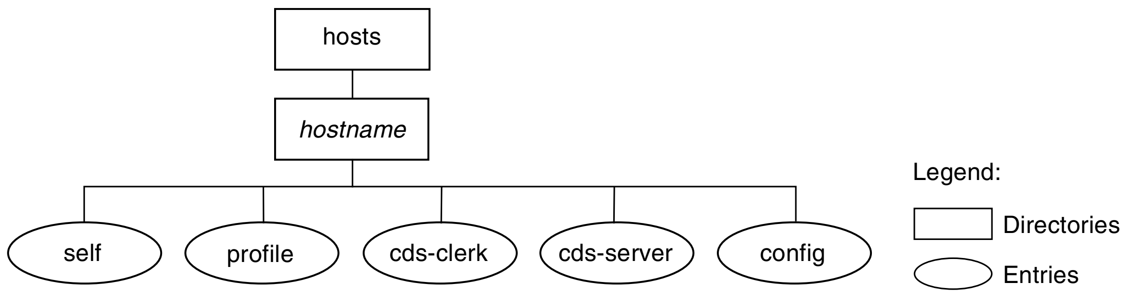


Figure 91. The HOSTS Directory in The CDS Namespace

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.
Initial Configuration ACL	
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. In multi-LAN cells, this is the profile in which the DTS global set entries are entered.

Initial Configuration ACL	
Object ACL	unauthenticated:r--t- user:*creator*:rwdtc group:subsys/dce/cds-admin:rwdtc group:subsys/dce/cds-server: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 RPC bindings of the following form: /.../*cellname*/hosts/*hostname*/self This object must 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r--t- user:*creator*:rwdtc group:subsys/dce/cds-admin:rwdtc group:subsys/dce/cds-server: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 cataloged here.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r--t--- user:*creator*:rwdtcia user:hosts/*hostname*/cds-server:rwdtcia user:hosts/*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 DTS local set entries are entered.
Initial Configuration ACL	
Object ACL	unauthenticated:r--t- user:*creator*:rwdtc group:subsys/dce/cds-admin:rwdtc group:subsys/dce/cds-server:rwdtc group:subsys/dce/dts-admin:rwdtc group:subsys/dce/dts-servers:rwdtc any_other:r--t-
Created by	DCE configuration

Name	/.:/*lclhostname*_ch
CDS Type	Object
CDS Class	CDS_Clearinghouse
Well known	No
Description	All clearinghouses are cataloged in the cell root. This name is only fixed for the first CDS server you configure. You can choose different names for any additional CDS servers you configure.
Initial Configuration ACL	
Object ACL	unauthenticated:r--t- group:subsys/dce/cds-admin:rwdtc group:subsys/dce/cds-server:rwdtc any_other:r--t-.
Created by	DCE 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 /.../*cellname*/subsys/dce/sec/rep_1. This is the junction into the Security namespace.
Initial Configuration ACL	

Object ACL	Default cell = /.../*cellname* unauthenticated:r--t- user:*creator*:rwdtc user:dce-rgy:rwdtc user:hosts/*rep_1_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. It contains the DCE subdirectory. 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 should be stored in directories below subsys.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r--t--- user:*creator*:rwdtcia user:hosts/*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

Name	/.:/hosts/*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 beneath the directory.
Initial Configuration ACL	

Object ACL	Default cell = /.../*cellname* unauthenticated:r--t--- user:*creator*:rwdtcia user:hosts/*hostname*/cds-server:rwdtcia user:hosts/*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	/.:/hosts/*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 used by the ACL editor to manage the ACL interface.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r--t- user:*creator*:rwdtc user:hosts/*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/*hostname*/cds-gda
CDS Type	Object
CDS class	RPC_Entry
Well known	No
Description	This entry contains the binding for a GDA server. This entry is used by the ACL editor to manage the ACL interface.
Initial Configuration ACL	

Object ACL	Default cell = /.../*cellname* unauthenticated:r--t- user:*creator*:rwdtc user:hosts/*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/*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 used by the ACL editor to manage the ACL interface.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r--t- user:*creator*:rwdtc user:hosts/*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/*hostname*/profile
CDS Type	Object
CDS class	RPC_Entry
Well known	No
Description	This is the default profile for host *hostname*. It must contain a default, which points (possibly indirectly) at /.:/cell-profile. Programs obtain this name using the dce_cf_profile_name_from_host() call.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r--t- user:*creator*:rwdtc user:hosts/*hostname*/self:rw-t- group:subsys/dce/cds-admin:rwcdt group:subsys/dce/cds-server:rwcdt any_other:r--t-.
Created by	DCE configuration

Name	/.:/hosts/*hostname*/self
CDS Type	Object
CDS class	RPC_Entry
Well known	Yes

Description	This entry contains a binding to the rpcd daemon on host *hostname*. The dce_cf_binding_entry_from_host() call returns either the name of this entry when handed a host name or the current host when a host name is not provided.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r--t- user:*creator*:rwdtc user:hosts/*hostname*/self:rwrdtc group:subsys/dce/cds-admin:rwdtc group:subsys/dce/cds-server:rwdtc any_other:r--t-.
Created by	DCE configuration

The CDS subsys Directory

Name	/.:/subsys/dce
CDS Type	Directory
Well known	No
Description	This directory contains DCE specific names.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r--t--- user:*creator*:rwdtcia user:hosts/*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.
Initial Configuration ACL	

Object ACL	Default cell = /.../*cellname* unauthenticated:r--t--- user:*creator*:rwdtcia user:hosts/*hostname*/cds-server:rwdtcia group:subsys/dce/cds-admin:rwdtcia group:subsys/dce/cds-server:rwdtcia group:subsys/dce/dfs-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-- 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/*hostname*/self. In addition, this group must have a single object UUID attached to it.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r--t- user:*creator*:rwdtc user:hosts/*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/sec
CDS Type	Directory
Well known	No
Description	This directory contains Security specific names.
Initial Configuration ACL	

Object ACL	Default cell = /.../*cellname* unauthenticated:r--t--- user:*creator*:rwdtcia user:hosts/*hostname*/cds-server:rwdtcia user:dce-rgy:rwdtci- user:hosts/*rep_1_hostname*/self:rwdtia 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--- user:hosts/*rep_1_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---.
Def_container ACL	Default cell = /.../*cellname* unauthenticated:r--t--- user:dce-rgy:rwdtci- user:hosts/*rep_1_hostname*/self:rwdtcia 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r--t- user:dce-rgy:rwdt- user:*creator*: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/dce/sec/rep_1
CDS Type	Object
CDS Class	RPC_Entry
Well known	No

Description	This is the server entry for a slave 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r--t- user:dce-rgy:rwdt- user:*creator*:rwdtc user:hosts/*rep_1_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

The Security Namespace

This section describes the entries in the Security namespace within the DCE cell namespace. The subdirectories and objects that comprise the Security namespace are **principal**, **group**, **org**, **policy**, **replist**, and **xattrschema**.

To view the ACLs on any of these namespace entries, you need to include the name of the Security junction. For example, the group name acct-admin is referenced as /.:/sec/group/acct-admin when you use the ACL Editor.

In contrast to the ACL Editor, the Registry Editor operates on a principal, group, or organization name without including /.:/sec and **principal**, **group**, or **org** as part of the name. To operate on the group acct-admin using the Registry Editor, you specify the domain **group** and the group name, acct-admin.

In the following subsections, descriptions of entries in an initial Security namespace are given. Included is the suggested UNIX user identifier (UNIX uid) or group identifier (UNIX gid) that they are assigned to. Some entries are assigned the next available identifier (starting with 100), so these may vary from cell to cell. They are indicated as *generated*.

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. This name is only used by the ACL Editor.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r----- user:*creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----.

Def_object ACL	Default cell = /.../*cellname* unauthenticated:r-t----- user:*creator*:rctDnfmM group_obj:r-t----- group:acct-admin:rctDnfmM other_obj:r-t----- any_other:r-------.
Def_container ACL	Default cell = /.../*cellname* 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 the organizations. This name is only used by the ACL editor.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r----- user:*creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----.
Def_object ACL	Default cell = /.../*cellname* unauthenticated:r-t----- user:*creator*:rctDnfmM group:acct-admin:rctDnfmM other_obj:r-t----- any_other:r-t-----.
Def_container ACL	Default cell = /.../*cellname* 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	This is the default organization.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r-t----- user:*creator*:rctDnfmM group:acct-admin:rctDnfmM other_obj:r-t----- any_other:r-t-----

UNIX org id	12
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's ACL controls the ability to set Security policies on a cell-wide basis.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r---- user:*creator*:rcmaA group:acct-admin:rcmaA other_obj:r---- any_other:r----.
Created by	DCE configuration

Name:	/.:/sec/replist
Well known	Yes. This name is not architecturally defined, but is defined by the implementation.
Description	This ACL controls access to the security server's replica list.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* user:*cell_admin*:cidmA- user:**hosts/hostname/self**:-i-m-I group:acct-admin:cidmA-
Created by	DCE configuration

Name:	/.:/sec/xattrschema
Well known	Yes. This name is not architecturally defined, but is defined by the implementation.
Description	This ACL controls access to extended registry attributes.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r--- user:*cell_admin*:rcidm 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 V1.0.
Description	This is the Security directory that holds all the principals. This name is only used by the ACL editor.
Initial Configuration ACL	

Object ACL	Default cell = /.../*cellname* unauthenticated:r----- user:*creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other_obj:r-----.
Def_object	Default cell = /.../*cellname* unauthenticated:r-------g user_obj:r---f--ug user:*creator*:rcDnfmaug group:acct-admin:rcDnfmaug other_obj:r-------g any_other:r--------.
Def_container	Default cell = /.../*cellname* unauthenticated:r----- user:*creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----.
Created by	Security configuration

The sec/group Directory

Name	/.:/sec/group/acct-admin
Security Type	Group
Well known	No
Description	This is the only group of principals that can create accounts.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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 with read access to kernel memory.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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	The default group for NFS** access (goes with user ID nobody).
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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 is a member of no group; the default group.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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
Security Type	Group Directory
Well known	Yes
Description	This directory contains DCE. (See /.:/subsys in the CDS namespace.)
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r----- user:*creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----.
Def_object ACL	Default cell = /.../*cellname* unauthenticated:r-t----- user:creator:rctDnfmM group_obj:r-t----- group:acct-admin:rctDnfmM other_obj:r-t----- any_other:r-t-----.

Def_container ACL	Default cell = /.../*cellname* 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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 with write access to terminals.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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

Name	/.:/sec/group/subsys/dce
Security Type	Group Directory
Well known	Yes
Description	This directory contains DCE required groups.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r----- user:*creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----.
Def_object ACL	Default cell = /.../*cellname* unauthenticated:r-t----- user:*creator*:rctDnfmM group_obj:r-rt----- group:acct-admin:rcitDnfmM other_obj:r-t----- any_other:r-t-----.
Def_container ACL	Default cell = /.../*cellname* 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
Security Type	Group
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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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/cds-server
Security Type	Group
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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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
Security Type	Group
Well known	No
Description	This is the DFS administrators' group. Members of this group have full permission to alter the DFS configuration within the cell.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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
Security Type	Group
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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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
Security Type	Group
Well known	Yes
Description	Abbreviated forms of the DFS server principals of all Fileset Database machines are listed in this group. The abbreviated form of a machine's DFS server principal stored in the group is of the form hosts/*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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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/group/subsys/dce/dskl-admin
Security Type	Group
Well known	No.
Description	This is the Diskless Service administrators' group.
Initial Configuration ACL	

Object ACL	Default cell = /.../*cellname* 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-admin
Security Type	Group
Well known	No.
Description	This is the DTS administrators' group. Members of this group have full permissions to administer DTS by adding servers and so forth.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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
Security Type	Group
Well known	Yes
Description	This is the group of DTS servers.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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
Security Type	Group
Well known	No
Description	This is the Security administrators' group. Members of this group have full permissions to administer the Security database.

Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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

Name:	/.:/sec/principal/bin
Well known	No
Description	The owner of system binaries.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r-------- user_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	The DCE cell administrator.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r-------- user_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 various daemons.
Initial Configuration ACL	

Object ACL	Default cell = /.../*cellname* 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
Security Type	Principal
Well known	Yes
Description	This is the architecturally defined principal name of the Privilege Server.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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
Security Type	Principal
Well known	Yes
Description	This is the architecturally defined principal name of the Registry Server.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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
Security Type	Principal Directory
Well known	No
Description	This directory contains .:/hosts/*hostname* directories.
Initial Configuration ACL	

Object ACL	Default cell = /.../*cellname* unauthenticated:r----- user:*creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----.
Def_object ACL	Default cell = /.../*cellname* unauthenticated:r--------- user_obj:r---f--ug user:*creator*:rcDnfmaug group:acct-admin:rcDnfmaug other_obj:r-------g any_other:r--------.
Def_container ACL	Default cell = /.../*cellname* 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 **/...**).
Security Type	Principal Directory
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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r----- user:*creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----.
Def_object ACL	Default cell = /.../*cellname* unauthenticated:r-------- user_obj:r---f--ug user:*creator*:rcDnfmaug group:acct-admin:rcDnfmaug other_obj:r-------g any_other:r--------.
Def_container ACL	Default cell = /.../*cellname* 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 **/.:**)
Security Type	Principal
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 /.: .
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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.
Initial Configuration ACL	

Object ACL	Default cell = /.../*cellname* 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 that is permitted to read devices but is not a superuser.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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.
Initial Configuration ACL	

Object ACL	Default cell = /.../*cellname* 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.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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

Name	/.:/sec/principal/hosts/*hostname*
Security Type	Principal Directory
Well known	No
Description	This directory contains Security principals for host *hostname*.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* unauthenticated:r----- user:*creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----.
Def_object ACL	Default cell = /.../*cellname* unauthenticated:r-------g user_obj:r---f--ug user:*creator*:rcDnfmaug group:acct-admin:rcDnfmaug other_obj:r-------g any_other:r--------.

Def_container ACL	Default cell = /.../*cellname* unauthenticated:r----- user:*creator*:rcidDn group:acct-admin:rcidDn other_obj:r----- any_other:r-----.
Created by	Security configuration

Name	/.:/sec/principal/hosts/*hostname*/cds-server
Security Type	Principal
Well known	No
Description	A CDS server on node *hostname* runs as this principal. This principal must be a member of /.:/subsys/dce/cds-server.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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	DCE configuration

Name	/.:/sec/principal/hosts/*hostname*/dfs-server
Security Type	Principal
Well known	No
Description	This is the principal name of DFS servers.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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	DCE configuration

Name	/.:/sec/principal/hosts/*hostname*/gda
Security Type	Principal
Well known	No
Description	The GDA on node *hostname* runs as this principal. This principal must be a member of /.:/subsys/dce/cds-servers.
Initial Configuration ACL	

Object ACL	Default cell = /.../*cellname* 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/*hostname*/self
Security Type	Principal
Well known	Yes
Description	This entry is the principal for the host *hostname*. The Security client daemon uses this principal. This can also be the principal that child processes of the init command use.
Initial Configuration ACL	
Object ACL	Default cell = /.../*cellname* 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	Security configuration

Appendix D. Valid Characters and Naming Rules for CDS

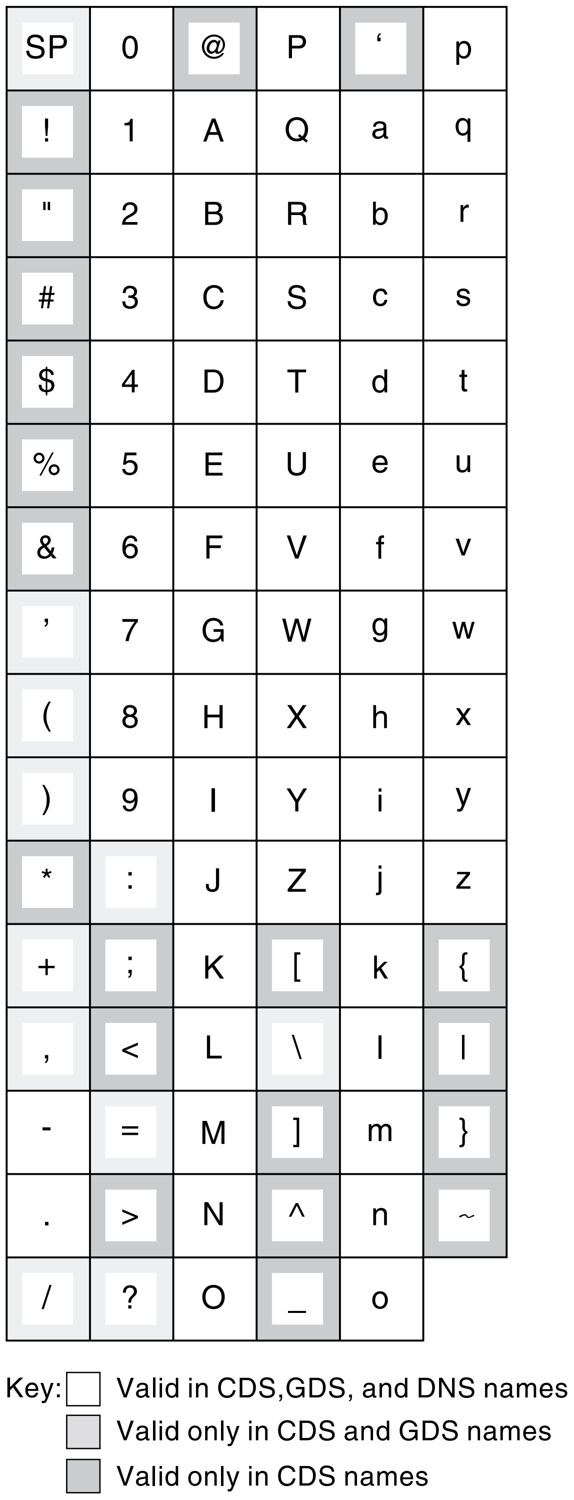
This appendix discusses the valid character sets for DCE Directory Service names as used by CDS interfaces. It also explains some characters that have special meaning and describes some restrictions and rules regarding case matching, syntax, and size limits. It is not a comprehensive reference for CDS, 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.

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 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 inter-operations 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 92 on page 530 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.



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

Key: □ Valid in CDS,GDS, and DNS names
░ Valid only in CDS and GDS names
▓ Valid only in CDS names

Figure 92. Valid Characters in CDS, GDS, and DNS Names

Although spaces are valid in both CDS and GDS names, a CDS simple name containing a space must be enclosed in quotation marks when you enter it through the CDS control program. Additional interface-specific rules are documented in the parts where they apply.

Metacharacters

Certain characters have special meaning to the directory services; these are known as **metacharacters**. Table 33 lists and explains the CDS, GDS, and DNS metacharacters.

Table 33. 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, acts as a wildcard, matching 0 or more characters.
	?	When used in the rightmost simple name of a name entered in a CDSCP show or list command, acts as a wildcard, matching exactly one character.
	\	Used where necessary in front of 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 a "/" (slash), a "," (comma), or an "=" (equal 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 usual characters within a name. For example, a back slash (\) cannot be used as anything but an escape character in GDS. You can use other metacharacters as usual characters in a name, provided that you escape them with the back slash metacharacter.

Additional Rules

Table 34 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 34 (Page 1 of 2). 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 the Valid Characters table.	a to z, A to Z, 0 to 9 plus .:, ' + - = () ? / and space.	a to z, A to Z, 0 to 9 plus . -
Metacha-racters	/ * ? \	/ , = \	.

Table 34 (Page 2 of 2). Summary of CDS, GDS, and DNS Characteristics

Characteristic	**CDS**	**GDS**	**DNS**
Restrictions	Simple names cannot contain a forward slash or a back slash character. The back slash can still be used as an escape character, but cannot be used as part of the simple name. That is, you cannot have "\\". First simple name following the global cell name (or /.: prefix) cannot contain an "=" (equal sign). When entering a name as part of a CDSCP show or list command, you must use a back slash (\) to escape any "*" (asterisk) or "?" (question mark) character in the rightmost simple name. Otherwise, the character is interpreted as a wildcard.	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 back slash (\) to escape a "/" (slash), a "," (comma), and an "=" (equal sign) when using them as anything other than metacharacters. Multiple, consecutive, unescaped occurrences of "/" (slashes), "," (commas), "=" (equal signs), and back slashes (\) are not allowed. Each attribute value assertion contains exactly one unescaped "=" (equal sign).	First character must be alphabetic. First and last characters cannot be a "." (dot) or a "-" (dash). Cell names in DNS must contain at least one dot; 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 rule says case-insensitive, alphabetic characters are converted to all lowercase characters. Spaces are removed regardless of case-matching rule.	Alphabetic characters are converted to all lowercase characters.
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.)

Maximum Name Sizes

Table 35 lists maximum sizes for Directory Service names. Note that the limits are implementation-specific, not architectural.

Table 35. Maximum Sizes of Directory Service Names

Name Type	Maximum Size
CDS simple name (character string between two slashes.)	255 characters
CDS full name (including global or local prefix, cell name, and slashes separating simple names.) **Note:** The server may or may not support the maximum size indicated here. Check the CDS documentation for the particular platform that offers the CDS server.	402 characters
GDS relative distinguished name	64 characters
GDS distinguished name	1024 characters
DNS relative name (character string between two dots)	64 characters
DNS fully qualified name (sum of all relative names)	255 characters

Appendix E. 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 Directory Service, every CDS attribute name and GDS attribute type label has a corresponding unique number called an **object identifier**.

CDS provides a method for translating between object identifiers and human-readable names. Users can enter names instead of object identifiers at the CDS control program interface. Also, the CDS control program displays the names rather than object identifiers in command output. CDS attribute names and their corresponding identifiers are stored in the file **/opt/dcelocal/etc/cds_attributes**. GDS attribute type labels and their corresponding identifiers are stored in the file **/opt/dcelocal/etc/cds_globalnames**.

This appendix describes the contents and usage of both the **cds_attributes** and **cds_globalnames** files and explains how application developers or Directory Service managers can update the files with the object identifiers of new attributes.

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). 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. For example, the 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.

CDS Attributes File

The **cds_attributes** file contains object identifiers for CDS attributes and object classes. 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
1.3.22.1.3.35   CDS_CHUpPointers        ReplicaPointer
                           .
                           .
                           .
```

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.

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.

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. 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. The 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_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.

Modifying the Files

When a programmer develops an application that uses the DCE 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 **/opt/dcelocal/etc/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 **/opt/dcelocal/etc/cds_globalnames** file and use the appropriate ASN.1 identifiers to describe the data type, syntax, and case-matching rules for the name.

Notes:

1. The **/opt/dcelocal/etc/cds_attributes**, **/opt/dcelocal/etc/cds_globalnames**, and **/opt/dcelocal/etc/xoischema** files must be in code page IBM-1047. If you port a script file from another platform, be sure that it is converted to code page IBM-1047.
2. When editing the cds_attributes and cds_globalnames files, do not leave any blank line or lines with unwanted characters. The presence of blank lines or lines with unwanted characters in these files will cause the CDS Clerk to hang.
3. If you modify the OID values for standard attributes in the cds_attributes and cds_globalnames files, you may encounter problems inter-operating with other Directory Service implementations.
4. If you modify the **cds_attributes** or **cds_globalnames** file, you must restart the CDS daemon.

Appendix F. DTS Extended BNF

This appendix defines the Distributed Time Service (DTS) syntax in extended Backus Naur format (BNF) notation.

DTS Format Rules

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 is:

```
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
;
```

Appendix G. Files Created and Used by mvsimpt and mvsexpt

This appendix lists all files created by or used as input to the cross-linking utilities, **mvsimpt** and **mvsexpt**. Use of these utilities is described in Chapter 41, "RACF Interoperability and Single Sign-on" on page 389.

Files Common to mvsimpt and mvsexpt

These files are created or used by **mvsimpt** and **mvsexpt**. Examples of the files are shown in "Examples of Files Created or Used by mvsimpt and mvsexpt" on page 543.

- **/opt/dcelocal/var/security/adm/DCEERS**

  The DCE error file is used by **mvsimpt** and **mvsexpt** to hold any errors that result from the processing of commands to the DCE Registry. The error file contains the original command entered and the resulting error messages. The error file can be used as an input file for **mvsimpt -p1**. To do this, after the error conditions have been corrected, edit the file to remove the error messages. The file is then specified using the command option **-r**. For example, to run pass one of **mvsimpt** with the error file, type:

  ```
  mvsimpt -p1 -r
  ```

  For an example of this file, see Figure 93 on page 543.

  Note: Because this file contains user passwords and the DCE administrator's password, it should be provided appropriate protection.

- **/opt/dcelocal/var/security/adm/PROCENTR**

  The Processed Entries file is used by **mvsimpt** and **mvsexpt** to keep track of users that have been previously processed. This file acts as a filter for successive invocations of pass one for **mvsimpt** and **mvsexpt** so that the commands generated are not repeated for users who have already been processed. For an example of this file, see Figure 94 on page 543. For layout and format of the file, see Figure 95 on page 543 and Table 36 on page 544.

- **/opt/dcelocal/dcecp/mvsimpt.dcp** and **/opt/dcelocal/dcecp/mvsexpt.dcp**

  The DCE Tool Command Language (Tcl) Scripts are programs used by **mvsimpt** and **mvsexpt** to process commands to the DCE Registry.

Files for mvsimpt

These files are used or created by **mvsimpt**. Examples of the files are shown in "Examples of Files Created or Used by mvsimpt" on page 545.

- **/opt/dcelocal/etc/IMPTVAR**

  The MVS Import variables file is used by **mvsimpt** as a primary input file to set the values of all variables that can be tailored by the administrator of the utilities. For an example of this file, see Figure 96 on page 545.

- **/opt/dcelocal/var/security/adm/RACFUNLD**

  The RACF Unload file is used by **mvsimpt** as its primary input file and contains the set of RACF users that the administrator needs to cross-link to DCE. For a description of how this file is created, see "Cross Linking Existing RACF Users who are New DCE Users" on page 400 For an example of this file, see Figure 97 on page 546. For layout and format of the file, see Figure 98 on page 546 and Table 37 on page 547.

There is also an alternate DCE Segments input file specified by the **-s** option of **mvsimpt** pass one. This file can be any HFS file with the content in the same form as the RACF unload file, **/opt/dcelocal/var/security/adm/RACFUNLD**, shown in Figure 97 on page 546.

- **/opt/dcelocal/var/security/adm/DCEWORK**

  The DCE Work file is created by **mvsimpt -p1** and contains the **dcecp** DCE administration commands necessary to add users to the DCE Registry. Then **mvsimpt -p2** processes this command file creating users in the DCE Registry. For an example of this file, see Figure 99 on page 547.

  Note: Because this file contains user passwords and the DCE administrator's password, it should be provided appropriate protection.

- **/opt/dcelocal/var/security/adm/DCENEW**

  The DCE New Accounts file is created by **mvsimpt -p2** and contains the users who were successfully enrolled in the DCE Registry. This file includes each user's initial password, which expires on first login to DCE. For an example of this file, see Figure 100 on page 547.

  Note: Because this file contains user passwords and the DCE administrator's password, it should be provided appropriate protection.

Files for mvsexpt

These files are created or used by **mvsexpt**. Examples of the files are shown in “Examples of Files Created or Used by mvsexpt” on page 548.

- **/opt/dcelocal/etc/EXPTVAR**

  The MVS Export variables file is used by **mvsexpt** as a primary input file to set the values of all variables that can be tailored by the administrator of the utilities. For an example of this file, see Figure 101 on page 548.

- **/opt/dcelocal/var/security/adm/RACFWORK**

  The RACF Work file is created by **mvsexpt -p1** and contains the RACF administration commands necessary to create or update users. **mvsexpt -p2** processes this command file to the RACF database. For an example of this file, see Figure 102 on page 550.

- **/opt/dcelocal/var/security/adm/RACFERS**

  The RACF error file is created by **mvsexpt -p2** and used by **mvsexpt** to hold any errors that result from the processing of commands to the RACF database. The error file contains the original command entered and the resulting error messages. For **mvsexpt -p2** the error file can be used as an input file. After the error conditions have been corrected, the file can be edited to remove the error messages. The file is then specified using the command option **-r**. For example, to run pass two of **mvsexpt** with the RACF error file, type:

  ```
  mvsexpt -p2 -r
  ```

  For an example of this file, see Figure 103 on page 550.

- **/opt/dcelocal/var/security/adm/RACFNEW**

  The RACF New Accounts file is created by **mvsexpt -p2** and contains the users who were successfully added to the RACF database. This file provides an information message for the administrator showing which users were created. For an example of this file, see Figure 104 on page 551.

- **/opt/dcelocal/var/security/adm/ASUIDMAP**

This file is the Application Support or similar identity mapping side file. Use of this file as part of Application Support can be eliminated by using the RACF database. This migration file is used as input to the **mvsexpt** utility by specifying, **mvsexpt -p1 -m**. See “Cross Linking Existing DCE Users who are Existing RACF Users” on page 407 for further information. For an example of this file, see Figure 105 on page 551. For the format of the file, see Figure 106 on page 551.

- **/opt/dcelocal/var/security/adm/PRNIDMAP**

  The principal identity mapping file is similar to the Application Support identity mapping side file. The principal identity mapping files need only contain DCE user principal IDs. **mvsexpt -p2 -u** takes the DCE user principal ID and generates an OS/390 ID. This OS/390 ID consists of the first one to seven valid OS/390 characters of the DCE principal ID. If no valid OS/390 ID can be generated this way, the utility creates an ID that consists of the prefix “DCE” and a four-digit random number. For an example of this file, see Figure 107 on page 552.

Examples of Files Created or Used by mvsimpt and mvsexpt

This section shows examples of the files used by both the **mvsimpt** and the **mvsexpt** utilities.

The /opt/dcelocal/var/security/adm/DCEERS file

```
user create Palmer -group Project -organization None -force -password nick341 -pwdvalid {no} -mypwd cell_adm_pw
Error: Registry object not found
```

Figure 93. The /opt/dcelocal/var/security/adm/DCEERS file

The /opt/dcelocal/var/security/adm/PROCENTR file

```
NICKLAUS                                             nicklaus    /.../dcecell1.endicott.ibm.com
PALMER    000000df-70e7-21ce-8e00-000000684530 Palmer      /.../dcecell1.endicott.ibm.com
NORMAN                                               GregNorman  /.../dcecell1.endicott.ibm.com
PRICE                                                Price       /.../dcecell1.endicott.ibm.com
```

Figure 94. The /opt/dcelocal/var/security/adm/PROCENTR file

Layout of the /opt/dcelocal/var/security/adm/PROCENTR file

```
PALMER    000000df-70e7-21ce-8e00-000000684530 Palmer      /.../dcecell1.endicott.ibm.com
|         |                                    |           |
|         |                                    |           |Home Cell Name
|         |                                    |DCE Principal Name
|         |UUID
|MVS ID
```

Figure 95. Layout of the /opt/dcelocal/var/security/adm/PROCENTR file

Format of the /opt/dcelocal/var/security/adm/PROCENTR file

Table 36. PROCENTR Data Record Format

Field Name	Type	Field Start	Field End	Comments
MVS ID	Char	1	8	MVS User ID.
UUID	Char	10	45	Universal Unique Identifier.
DCE Principal Name	Char	47	1069	DCE User Principal Name associated with the MVS ID.
Home Cell Name	Char	1071	2093	DCE Cell Name

Note: Record format is the same as RACFUNLD except that the field **Type** has been removed.

Examples of Files Created or Used by mvsimpt

This section shows examples of the files used by **mvsimpt**.

The /opt/dcelocal/etc/IMPTVAR file

```
/*------------------------------------------------------------------*/
/*-Variables file for mvsimpt rexx exec                            -*/
/*-/opt/dcelocal/etc/IMPTVAR                                       -*/
/*------------------------------------------------------------------*/

/*------------------------------------------------------------------*/
/*-Default DCE Group                                               -*/
/*------------------------------------------------------------------*/
DCEGRP   Project

/*------------------------------------------------------------------*/
/*-Default DCE Organization                                        -*/
/*------------------------------------------------------------------*/
DCEORG   None

/*------------------------------------------------------------------*/
/*-Default cell-name                                               -*/
/*-Use a global name                                               -*/
/*------------------------------------------------------------------*/
CELLNAM  /.../dcecell11.endicott.ibm.com

/*------------------------------------------------------------------*/
/*-Default RACF homecell                                           -*/
/*-Use a global name                                               -*/
/*------------------------------------------------------------------*/
DEFCELL /.../dcecell11.endicott.ibm.com

/*------------------------------------------------------------------*/
/*-RACF Unload file                                                -*/
/*------------------------------------------------------------------*/
RACFUNLD /opt/dcelocal/var/security/adm/RACFUNLD

/*------------------------------------------------------------------*/
/*-Processed Entries file                                          -*/
/*------------------------------------------------------------------*/
PROCENTR /opt/dcelocal/var/security/adm/PROCENTR
```

Figure 96 (Part 1 of 2). The /opt/dcelocal/etc/IMPTVAR file

```
/*------------------------------------------------------------------*/
/*-DCE Work file                                                   -*/
/*------------------------------------------------------------------*/
DCEWORK  /opt/dcelocal/var/security/adm/DCEWORK

/*------------------------------------------------------------------*/
/*-DCE New Accounts file                                           -*/
/*------------------------------------------------------------------*/
DCENEW  /opt/dcelocal/var/security/adm/DCENEW

/*------------------------------------------------------------------*/
/*-DCE Error file                                                  -*/
/*------------------------------------------------------------------*/
DCEERS  /opt/dcelocal/var/security/adm/DCEERS

/*------------------------------------------------------------------*/
/*-DCE Tcl script file                                             -*/
/*------------------------------------------------------------------*/
DCETCL /opt/dcelocal/dcecp/mvsimpt.dcp
```

Figure 96 (Part 2 of 2). The /opt/dcelocal/etc/IMPTVAR file

The /opt/dcelocal/var/security/adm/RACFUNLD file

```
0290 NICKLAUS                                      nicklaus    /.../dcecell1.endicott.ibm.com
0290 PALMER   000000df-70e7-21ce-8e00-000000684530 Palmer      /.../dcecell1.endicott.ibm.com 100b49ae-b7c4-11ce-93fd-000000065630
0290 NORMAN                                        GregNorman
0290 PRICE                                         Price       /.../dcecell1.endicott.ibm.com
```

Figure 97. The /opt/dcelocal/var/security/adm/RACFUNLD file

Layout of the /opt/dcelocal/var/security/adm/RACFUNLD file

```
0290 PALMER 000000df-70e7-21ce-8e00-000000684530 Palmer /.../dcecell1.endicott.ibm.com 100b49ae-b7c4-11ce-93fd-000000065630 YES
|    |      |                                    |      |                              |                                    |
|    |      |                                    |      |Home Cell Name                |Home Cell UUID                      |
|    |      |                                    |DCE Principal Name                                                        |
|    |      |UUID                                                                                         DCE Autologin Flag|
|    |MVS ID
|
|Record Type
```

Figure 98. Layout of the /opt/dcelocal/var/security/adm/RACFUNLD file

Format of the /opt/dcelocal/var/security/adm/RACFUNLD file

Table 37. RACFUNLD Data Record Format

Field Name	Type	Field Start	Field End	Comments
Type	Char	1	4	Record type of the User DCE Data record (0290).
MVS ID	Char	6	13	MVS User id.
UUID	Char	15	50	User Principal Universal Unique Identifier(UUID)
DCE Principal Name	Char	52	1074	DCE User Principal Name associated with the MVS id.
Home Cell Name	Char	1076	2098	DCE Cell Name
Home Cell UUID	Char	2100	2135	DCE Cell UUID
DCE Autologin	Char	2137	2140	DCE Automatic Login Flag (YES/NO)

The /opt/dcelocal/var/security/adm/DCEWORK file

```
user create nicklaus -group Project -organization None -force -password nick341 -pwdvalid {no} -mypwd cell_adm_pw
user create GregNorman -group Project -organization None -force -password Greg592 -pwdvalid {no} -mypwd cell_adm_pw
user create Price -group Project -organization None -force -password Pric45 -pwdvalid {no} -mypwd cell_adm_pw
```

Figure 99. The /opt/dcelocal/var/security/adm/DCEWORK file

The /opt/dcelocal/var/security/adm/DCENEW file

```
user create nicklaus -group Project -organization None -force -password nick341 -pwdvalid {no} -mypwd cell_adm_pw
New user created in DCE Registry

user create GregNorman -group Project -organization None -force -password Greg592 -pwdvalid {no} -mypwd cell_adm_pw
New user created in DCE Registry

user create Price -group Project -organization None -force -password Pric45 -pwdvalid {no} -mypwd cell_adm_pw
New user created in DCE Registry
```

Figure 100. The /opt/dcelocal/var/security/adm/DCENEW file

Examples of Files Created or Used by mvsexpt

This section shows examples of the files used by **mvsexpt**.

The /opt/dcelocal/etc/EXPTVAR file

```
/*-------------------------------------------------------------------*/
/*-Variables file for mvsexpt rexx exec                             -*/
/*-/opt/dcelocal/etc/EXPTVAR                                        -*/
/*-------------------------------------------------------------------*/

/*-------------------------------------------------------------------*/
/*-RACF OMVS Segment UID field                                      -*/
/*-Starting value for UID                                           -*/
/*-incremented by 1 for each ADDUSER command                        -*/
/*-------------------------------------------------------------------*/
UIDVAR   99

/*-------------------------------------------------------------------*/
/*-RACF OMVS Segment PROGRAM field                                  -*/
/*-Program name variable                                            -*/
/*-------------------------------------------------------------------*/
PROVAR  /bin/sh

/*-------------------------------------------------------------------*/
/*-Used to determine if the RACF command should include HOMECELL    -*/
/*-YES = include HOMECELL parameter                                 -*/
/*-NO  = DO NOT include HOMECELL parameter                          -*/
/*-------------------------------------------------------------------*/
HOMECELL YES

/*-------------------------------------------------------------------*/
/*-Used to determine if the RACF database should include HOMEUUID   -*/
/*-------------------------------------------------------------------*/
HOMEUUID YES

/*-------------------------------------------------------------------*/
/*-Used to determine if the RACF command should include AUTOLOGIN   -*/
/*-YES = include AUTOLOGIN parameter                                -*/
/*-NO  = DO NOT include AUTOLOGIN parameter                         -*/
/*-------------------------------------------------------------------*/
AUTOLOGIN YES

/*-------------------------------------------------------------------*/
/*-Default RACF homecell                                            -*/
/*-Use a global name                                                -*/
/*-------------------------------------------------------------------*/
DEFCELL /.../dcecell11.endicott.ibm.com
```

Figure 101 (Part 1 of 2). The /opt/dcelocal/etc/EXPTVAR file

```
/*------------------------------------------------------------------*/
/*-Default cell-name                                               -*/
/*-Use a global name                                               -*/
/*------------------------------------------------------------------*/
CELLNAM  /.../dcecell11.endicott.ibm.com

/*------------------------------------------------------------------*/
/*-RACF Work file                                                  -*/
/*------------------------------------------------------------------*/
RACFWORK  /opt/dcelocal/var/security/adm/RACFWORK

/*------------------------------------------------------------------*/
/*-RACF Error file                                                 -*/
/*------------------------------------------------------------------*/
RACFERS   /opt/dcelocal/var/security/adm/RACFERS

/*------------------------------------------------------------------*/
/*-RACF New Accounts file                                          -*/
/*------------------------------------------------------------------*/
RACFNEW  /opt/dcelocal/var/security/adm/RACFNEW

/*------------------------------------------------------------------*/
/*-DCE Error file                                                  -*/
/*------------------------------------------------------------------*/
DCEERS  /opt/dcelocal/var/security/adm/DCEERS

/*------------------------------------------------------------------*/
/*-Processed Entries file                                          -*/
/*------------------------------------------------------------------*/
PROCENTR  /opt/dcelocal/var/security/adm/PROCENTR

/*------------------------------------------------------------------*/
/*-AS-DCE ID Mapping file                                          -*/
/*------------------------------------------------------------------*/
ASUIDMAP  /opt/dcelocal/var/security/adm/ASUIDMAP

/*------------------------------------------------------------------*/
/*-Principal Mapping file                                          -*/
/*------------------------------------------------------------------*/
PRNIDMAP  /opt/dcelocal/var/security/adm/PRNIDMAP

/*------------------------------------------------------------------*/
/*-DCE Tcl script file                                             -*/
/*------------------------------------------------------------------*/
DCETCL  /opt/dcelocal/dcecp/mvsexpt.dcp
```

Figure 101 (Part 2 of 2). The /opt/dcelocal/etc/EXPTVAR file

The /opt/dcelocal/var/security/adm/RACFWORK file

```
ADDUSER COUPLES
 DCE(DCENAME('couples') UUID(000000df-70e7-21ce-8e00-000000684531)
 HOMECELL('/.../dcecell1.endicott.ibm.com')
 HOMEUUID(100b49ae-b7c4-11ce-93fd-000000065630)
 AUTOLOGIN(YES))
 OMVS(UID(99) HOME(/u/couples) PROGRAM(/bin/sh))

ALTUSER NICKLAUS
 DCE(DCENAME('nicklaus')
 UUID(000000df-70e7-21ce-8e00-000000684525)
 HOMECELL('/.../dcecell1.endicott.ibm.com')
 HOMEUUID(100b49ae-b7c4-11ce-93fd-000000065630)
 AUTOLOGIN(YES))

ALTUSER NORMAN
 DCE(DCENAME('GregNorman')
 UUID(000000df-70e7-21ce-8e00-000000684526)
 HOMECELL('/.../dcecell1.endicott.ibm.com')
 HOMEUUID(100b49ae-b7c4-11ce-93fd-000000065630)
 AUTOLOGIN(YES))

ALTUSER PRICE
 DCE(DCENAME('Price')
 UUID(000000df-70e7-21ce-8e00-000000684523)
 HOMECELL('/.../dcecell1.endicott.ibm.com')
 HOMEUUID(100b49ae-b7c4-11ce-93fd-000000065630)
 AUTOLOGIN(YES))
```

Figure 102. The /opt/dcelocal/var/security/adm/RACFWORK file

The /opt/dcelocal/var/security/adm/RACFERS file

```
ADDUSER COUPLES
 DCE(DCENAME('couples') UUID(000000df-70e7-21ce-8e00-000000684531)
 HOMECELL('/.../dcecell1.endicott.ibm.com')
 HOMEUUID(100b49ae-b7c4-11ce-93fd-000000065630)
 AUTOLOGIN(YES))
 OMVS(UID(99) HOME(/u/couples) PROGRAM(/bin/sh))
*Command failure message returned from RACF*

ALTUSER NICKLAUS
 DCE(DCENAME('nicklaus')
 UUID(000000df-70e7-21ce-8e00-000000684525)
 HOMECELL('/.../dcecell1.endicott.ibm.com'))
 HOMEUUID(100b49ae-b7c4-11ce-93fd-000000065630)
 AUTOLOGIN(YES))
*Command failure message returned from RACF*
```

Figure 103. The /opt/dcelocal/var/security/adm/RACFERS file

The /opt/dcelocal/var/security/adm/RACFNEW file

```
New RACF account created for MVS user: COUPLES
```

Figure 104. The /opt/dcelocal/var/security/adm/RACFNEW file

The /opt/dcelocal/var/security/adm/ASUIDMAP file

```
Palmer
PALMER
/.:/AS/server_1

nicklaus
NICKLAUS

GregNorman
GREGNORM
/.:/AS/server_2

Price
PRICE
```

Figure 105. The /opt/dcelocal/var/security/adm/ASUIDMAP file

Format of /opt/dcelocal/var/security/adm/ASUIDMAP file

```
Palmer -------------->DCE-user-ID
PALMER -------------->MVS-user-ID
/.:/AS/server_1 ----->Application-support-server-name

nicklaus ------------>DCE-user-ID
NICKLAUS ------------>MVS-user-ID

GregNorman ---------->DCE-user-ID
GREGNORM ------------>MVS-user-ID
/.:/AS/server_2 ----->Application-support-server-name

Price --------------->DCE-user-ID
PRICE --------------->MVS-user-ID
```

Figure 106. The Identity Mapping File. File can consist of just DCE user IDs to be associated with MVS user IDs or DCE user IDs to be associated with MVS user IDs and Application Support servers. These two types of entries can be in any order but a blank line is required between them.

The /opt/dcelocal/var/security/adm/PRNIDMAP file

```
Palmer

nicklaus
NICKLAUS

GregNorman
NORMAN

Price
```

Figure 107. The /opt/dcelocal/var/security/adm/PRNIDMAP file

Format of the /opt/dcelocal/var/security/adm/PRNIDMAP file

```
Palmer ---------->DCE-user-ID

nicklaus  ------->DCE-user-ID
NICKLAUS  ------->MVS-user-ID

GregNorman ------>DCE-user-ID
NORMAN ---------->MVS-user-ID

Price ----------->DCE-user-ID
```

Figure 108. The Principal Mapping File. File can consist of just DCE user IDs or DCE user IDs to be associated with MVS user IDs. These two types of entries can be in any order but a blank line is required between them.

Appendix H. DCE Security Administration Files

This section describes DCE Security files for system administration and provides related information. The topics cover:

- **aud_audit_events** The auditable events for the Audit service.
- **dts_audit_events** The auditable events for the time services.
- **sec_audit_events** The auditable events for the security services.

Auditable Events for the Audit Services

The OS/390 DCE provides programs for auditing events the are significant to the Audit Service. Among these events are:

- Administrative operations

  These are subdivided into **modify** and **query** operations.

- Filter operations

  These are subdivided into **modify** and **query** operations.

Event class definitions, together with filters, control the auditing execution at these code points. Filters can be updated dynamically. Filter files are maintained by a per-host audit daemon, and are shared among all the audit clients on the same host. The **dcecp** command interface program is used for maintaining the filters. The **dcecp** command is executable by all users and system administrators. The control on who is allowed to modify filters is done through the Audit daemon's ACL (Access Control List).

Administrative Operations: The **dce_audit_admin_modify** and **dce_audit_admin_query** event classes lump together the administrative operations that are performed on the Audit daemon.

The **dce_audit_admin_modify** event class has the following events that modify the operation of the Audit daemon:

EVT_MODIFY_STATE	Enables or disables the Audit daemon for logging.
EVT_MODIFY_SSTRATEGY	Modifies storage strategy. This can be any of the following: • **Save**. If the trail is full, it is backed up and renamed with a timestamp then writes on the original trail again. • **Wrap**. If the trail is full, goes back to the beginning of the file, overwriting previously written records.
EVT_REWIND	Rewinds the Audit daemon's central trail file.
EVT_STOP	Stops the Audit daemon.

The following are the audit code points in the Audit Service interfaces, with their Event Types, Event Classes, and any Event-Specific Information. The numbers given in parentheses are decimal and their equivalent hexadecimal values.

Event Type (Event Number, Event Classes)
    EVT_MODIFY_STATE (774 (0x306), dce_audit_admin_modify)

Event-Specific Information
    None

Event Type (Event Number, Event Classes)
EVT_MODIFY_SSTRATEGY (773 (0x305), dce_audit_admin_modify)

Event-Specific Information
None

Event Type (Event Number, Event Classes)
EVT_REWIND (775 (0x307), dce_audit_admin_modify)

Event-Specific Information
None

Event Type (Event Number, Event Classes)
EVT_STOP (776 (0x308), dce_audit_admin_modify)

Event-Specific Information
None

The **dce_audit_admin_query** event class has two events:

- EVT_SHOW_SSTRATEGY - Shows the storage strategy.
- EVT_SHOW_STATE - Shows the state of the Audit daemon.

Following are the details of this event class:

Event Type (Event Number, Event Classes)
EVT_SHOW_SSTRATEGY (777 (0x309), dce_audit_admin_query)

Event-Specific Information
None

Event Type (Event Number, Event Classes)
EVT_SHOW_STATE (778 (0x30a), dce_audit_admin_query)

Event-Specific Information
None

Filter Operations: The **dce_audit_filter_modify** and **dce_audit_filter_query** event classes are the filter operations that the Audit daemon handles.

The **dce_audit_filter_modify** event class has the following events:

- EVT_ADD_FILTER - Adds a filter.
- EVT_DELETE_FILTER - Removes all guides for a specific subject.
- EVT_REMOVE_FILTER - Removes a specific guide for a specific subject.

Following are the details of this event class:

Event Type (Event Number, Event Classes)
EVT_ADD_FILTER (771 (0x303), dce_audit_filter_modify)

Event-Specific Information
None.

Event Type (Event Number, Event Classes)
EVT_DELETE_FILTER (768 (0x300), dce_audit_filter_modify)

Event-Specific Information
None.

Event Type (Event Number, Event Classes)
EVT_REMOVE_FILTER (772 (0x304), dce_audit_filter_modify)

Event-Specific Information
None.

The **dce_audit_filter_query** contains two events:

- EVT_LIST_FILTER - Lists all subjects that have filters.
- EVT_SHOW_FILTER - Shows all filters for a specific subject.

Following are the details of this event class.

Event Type (Event Number, Event Classes)
EVT_LIST_FILTER (770 (0x302), dce_audit_filter_query)

Event-Specific Information
None.

Event Type (Event Number, Event Classes)
EVT_SHOW_FILTER (769 (0x301), dce_audit_filter_query)

Event-Specific Information

```
unsigned 32      esl_type
char *      subject_name  /* in record when event
                                        selection is not world*/
```

Related Information

Commands:

- **dcecp**

Files:

dce_audit_admin_modify	**dce_audit_filter_query**
dce_audit_admin_query	**dce_audit_filter_modify**

Auditable Events for the Time Services

The Audit Service allows for the auditing of certain security-significant events in the Time Server. Among these events are:

- Time Service processes
- Clock readings
- Global-set membership (in the Cell Service Profile)
- Time Service attributes

Event class definitions, together with filters, control the auditing execution at these code points. Filters can be updated dynamically. Filter files are maintained by a per-host audit daemon, and are shared among all the audit clients on the same host. The **dcecp** command interface program is used for maintaining the filters. The **dcecp** command is executable by all users and system administrators. The control on who is allowed to modify filters is done through audit daemon's ACL, which maintains the filters.

The Time Server RPC interfaces that manage the Time Service and request and provide the time include **time_control**, **time_service**, **gbl_time_service**, and **time_provider**.

The following are the audit code points in these Time Service interfaces, with their Event Types, Event Classes, and any Event-Specific Information. The numbers given in parentheses are decimal, followed by the equivalent hexadecimal value.

Control Interface (time_control) Operations: The **CreateCmd()** operation creates the Time Service as a server or a clerk. The caller must have write access to the management interface.

Event Type (Event Number, Event Classes)
 EVT_CREATE_CMD (512 (0x200), dce_dts_mgt_modify)

Event-Specific Information
```
signed32  servType
```

The **DeleteCmd()** operation deletes the Time Service entity from the system where the command is entered. This command stops the process. The caller must have write access to the management interface.

Event Type (Event Number, Event Classes)
 EVT_DELETE_CMD (513 (0x201), dce_dts_mgt_modify)

Event-Specific Information
 None

The **EnableCmd()** operation starts the DTS entity on the local node. This command makes the server available to the network. The *clockSet* argument tells the Time Service whether to set the clock after the first synchronization. The caller must have write access to the management interface.

Event Type (Event Number, Event Classes)
 EVT_ENABLE_CMD (514 (0x202), dce_dts._mgt_modify)

Event-Specific Information
```
signed32  clockSet
```

The **DisableCmd** operation disables the Time Service by making it unavailable to the network. In the case of servers, it makes it unavailable to the RPC client trying to talk to it. For clerks, it stops synchronizing with servers. The caller must have write access to the management interface.

Event Type (Event Number, Event Classes)
 EVT_DISABLE_CMD (515 (0x203), dce_dts_mgt_modify)

Event-Specific Information
 None

The **UpdateCmd()** operation gradually adjusts the clock on the local node to the specified time. The caller must have write access to the management interface.

Event Type (Event Number, Event Classes)
 EVT_UPDATE_CMD (516 (0x204), dce_dts_synch)

Event-Specific Information
```
utc_t  old_time
utc_t  new_time
```

The **ChangeCmd** operation changes the epoch number on the server and optionally sets the time to a new time. These values are passed in the argument *changeDir*. The caller must have write access to the management interface.

Event Type (Event Number, Event Classes)
 EVT_CHANGE_CMD (517 (0x205), dce_dts_synch)

Event-Specific Information

```
signed32   old_epoch
signed32   new_epoch
utc_t      old_time
utc_t      new_time
```

The **SynchronizeCmd()** operation causes the Time Service to synchronize immediately. If the argument *clockSet* is true, the clock is set to the new value after a synchronization. The caller must have write access to the management interface.

Event Type (Event Number, Event Classes)
EVT_SYNCHRONIZE_CMD (518 (0x206), dce_dts_synch)

Event-Specific Information

```
signed32  setClock
```

The **AdvertiseCm()** operation adds (advertises) this Time Server node as a member of the global set in the Cell Services Profile. The caller must have write access to the management interface.

Event Type (Event Number, Event Classes)
EVT_ADVERTISE_CMD (519 (0x207), dce_dts_mgt_modify)

Event-Specific Information
None

The **UnadvertiseCmd()** operation removes (unadvertises) this Time Server node as a member of the set of global servers in the Cell Services profile. The caller must have write access to the management interface.

Event Type (Event Number, Event Classes)
EVT_UNADVERTISE_CMD (520 (0x208), dce_dts_mgt_modify)

Event-Specific Information
None

The **SetDefaultCmd()** operation, when an attribute with no accompanying value is passed, sets an attribute to its default value. The attribute type is passed in the *setAttr* argument. The caller must have write access to the management interface.

Event Type (Event Number, Event Classes)
EVT_SET_DEFAULT_CMD (521 (0x209), dce_dts_mgt_modify)

Event-Specific Information

```
byte  useDefault
signed32  attribute
```

The **SetAttrCmd()** operation, when an attribute and an accompanying value is passed, sets an attribute to a value given. The attribute type is passed in *setAttr* argument and the attribute value in *AttrValue* argument. The caller must have write access to the management interface.

Event Type (Event Number, Event Classes)
EVT_SET_ATTR_CMD (522 (0x20A), dce_dts_mgt_modify)

Event-Specific Information

```
signed32  attribute
signed32  attribute_type
```

The **ShowAttrCmd()** operation, when passed an attribute name, queries the Time Service for the attribute's value. The attribute value is passed back in the argument *attrValue*. The caller must have read access to the management interface.

Event Type (Event Number, Event Classes)
EVT_SHOW_ATTR_CMD (523 (0x20B), dce_dts_mgt_query)

Event-Specific Information
```
signed32  attribute
signed32  attribute_type
```

The **ShowAllCharsCmd()** operation, when not passed a group name with the **all** value, queries the Time Service for the values of all the characteristic attributes and values. The caller must have read access to the management interface.

Event Type (Event Number, Event Classes)
EVT_SHOW_ALL_CHARS_CMD (524 (0x20C), dce_dts_mgt_query)

Event-Specific Information
None

The **ShowAllStatusCmd()** operation, when passed the **all status** value, queries the Time Service for the values of all the status attributes. The caller must have read access to the management interface.

Event Type (Event Number, Event Classes)
EVT_SHOW_ALL_STATUS_CMD (525 (0x20D), dce_dts_mgt_query)

Event-Specific Information
None

The **ShowAllCntrsCmd()** operation, when passed the **all counters** value, queries the Time Service for the values of all the counters. The caller must have read access to the management interface.

Event Type (Event Number, Event Classes)
EVT_SHOW_ALL_CNTRS_CMD (526 (0x20E), dce_dts_mgt_query)

Event-Specific Information
None

The **ShowLocServersCmd()** operation, when passed the **local servers** value, queries the Time Service for the servers in the local set. A variable conformant array is used to return the set of local servers available. The size of the array transmitted over RPC is determined at run-time. The caller must have read access to the management interface.

Event Type (Event Number, Event Classes)
EVT_SHOW_LOC_SERVERS_CMD (527 (0x20F), dce_dts_mgt_query)

Event-Specific Information
None

The **ShowGblServersCmd()** operation, when passed the **global servers** value, queries the Time Service for the servers in the global set. A variable conformant array returns the set of global servers available. The caller must have read access to the management interface.

Event Type (Event Number, Event Classes)
EVT_SHOW_GBL_SERVERS_CMD (528 (0x210), dce_dts_mgt_query)

Event-Specific Information
None

Time Provider Interface (time_provider) Operations: Auditable events in the RPC-based Time Provider Program (TPP) interfaces are defined here. These events are called by a Time Service daemon running as a server (in this case it makes an RPC client call to the TPP server).

The **ContactProvider()** operation sends initial contact message to the TPP. The TPP server responds with a control message. This operation may cause modification of the time server's (not the provider's) clock and should be defined to be an auditable event in the time server. There is no access control in the provider for this operation, but the integrity of the messages is protected.

Event Type (Event Number, Event Classes)
  EVT_CONTACT_PROVIDER (529 (0x211), dce_dts_time_provider)

Event-Specific Information
  None

The **ServerRequestProviderTime()** operation has the client send a request to the TPP for times. The TPP server responds with an array of time stamps obtained by querying the Time Provider hardware that it polls. There is no access control in the Time Provider for this operation, but the integrity of the message is protected.

Event Type (Event Number, Event Classes)
  EVT_REQUEST_PROVIDER_TIME (530 (0x212), dce_dts_time_provider)

Event-Specific Information
  None

Related Information

Commands:

- **advertise**
- **aud**
- **audfilter**
- **change**
- **create**
- **dcecp**
- **delete**
- **disable**
- **dts_intro**
- **dtsd**
- **enable**
- **exit**
- **help**
- **quit**
- **set**
- **show**
- **synchronize**
- **unadvertise**
- **update**

Files:

- **dce_dts_mgt_modify**
- **dce_dts_mgt_query**
- **dce_dts_synch**
- **dce_dts_time_provider**

Auditable Events for the Security Service

The Audit Service allows for the auditing of certain security-significant events in the OS/390 Security Server. Among these events are:

- Attempts at invoking Authentication Server/Ticket-granting Server/Privilege Server (AS/TGS/PS) operations.
- Deletion of OS/390 Security Server objects, including
  - ACLs
  - accounts
  - pgo items
  - registry properties
  - registry/organization policies
  - registry master key
- Attempts at invoking an operation that modifies OS/390 Security Server objects or updates an ACL.
- Attempts at invoking operations that involve access control.
- Failed client responses to the server's challenge, detected replays and incorrect ticket requests.
- The usage of cryptographic keys in the RPC runtime.
- Attempts at changing the maintenance/operation states of the registry server.

Event class definitions, together with filters, control the auditing execution at these code points. Filters can be updated dynamically. Filter files are maintained by a per-host audit daemon, and are shared among all the audit clients on the same host. The **dcecp** command interface program is used for maintaining the filters. The **dcecp** command is executable by all users and system administrators. The control on who is allowed to modify filters is done through the audit daemon's ACL, which maintains the filters.

OS/390 Security Server RPC interfaces include **krb5rpc**, **rdaclif**, **rdacliftmp**, **rpriv**, **rs_acct**, **rs_query**, **rs_rpladmn**, **rs_update**, **rsec_cert**, and **secidmap**. All the RPC interfaces are offered using the **rpc_c_authn_dce_secret** authentication service. The OS/390 Security Server's RPC runtime uses **dce-rgy** as its authentication identity. Within the same process, the security server's UDP/IP interface provides Kerberos AS/TGS functions, with **krbtgt/cell_name** as its authentication identity.

The following are the audit code points in these Security Service interfaces, with their Event Types, Event Classes, and any Event-Specific Information. The numbers given in parentheses are decimal, followed by the equivalent hexadecimal value.

Authentication Interface (krb5rpc) Operations: The **rsec_krb5rpc_sendto_kdc()** function is an RPC interface operation for accessing Kerberos AS/TGS services. Ticket-granting tickets and application tickets are requested and returned. There is no access control on this interface other than what is within the Kerberos Ticket-granting mechanism itself; that is, the TGS request verification.

Event Type (Event Number, Event Classes)
: **AS_Request (257 (0x101), dce_sec_authent)**

Event-Specific Information

```
long_int  kdc_error_code
boolean  account_disabled
```

Event Type (Event Number, Event Classes)
: **TGS_TicketReq (258 (0x102), dce_sec_authent)**

Event-Specific Information
```
char  *server_name
long_int  kdc_error_code
```

Event Type (Event Number, Event Classes)
: **TGS_RenewReq (259 (0x103), dce_sec_authent)**

Event-Specific Information
```
char  *server_name
long_int  kdc_error_code
```

Event Type (Event Number, Event Classes)
: **TGS_ValidateReq (260 (0x104), dce_sec_authent)**

Event-Specific Information
```
char  *server_name
long_int  kdc_error_code
```

DACL Management Interface (rdaclif) Operations: The **rdacl_lookup()** operation retrieves an ACL of an object in the OS/390 Security Server. Review of ACL associated with an object in OS/390 Security Server is allowed if the caller has any access to the object.

Event Type (Event Number, Event Classes)
: **ACL_Lookup (261 (0x105), dce_sec_control, dce_sec_query)**

Event-Specific Information
```
char    *component_name
uuid_t     manager_type
ulong_int     acl_type
```

The **rdacl_replace()** operation replaces the ACL of an object in the OS/390 Security Server. The client must have the **sec_acl_perm_owner** permission for the update to be carried out.

Event Type (Event Number, Event Classes)
: **ACL_Replace (262 (0x106), dce_sec_control, dce_sec_modify)**

Event-Specific Information
```
char    *component_name
uuid_t     manager_type
ulong_int     acl_type
```

The **rdacl_get_access()** operation determines the caller's access to a specified object. This call is authorized if the caller has any access to the object.

Event Type (Event Number, Event Classes)
: **ACL_GetAccess (263 (0x107), dce_sec_control, dce_sec_query)**

Event-Specific Information
```
char    *component_name
uuid_t     manager_type
ulong_int     requested_permissions
```

The **rdacl_test_access()** operation determines if the caller has the requested access. The return value of the call indicates whether the caller has the requested access to the object.

Event Type (Event Number, Event Classes)
: **ACL_TestAccess (264 (0x108), dce_sec_control, dce_sec_query)**

Event-Specific Information
```
char    *component_name
uuid_t     manager_type
ulong_int     requested_permissions
```

The **rdacl_get_manager_types()** operation lists the types (UUIDs) of ACLs protecting an object. The caller must have some permissions on the object for each of the manager types that is defined for the object. Otherwise, no manager type is returned.

Event Type (Event Number, Event Classes)
ACL_GetMgrTypes (266 (0x10A), dce_sec_control, dce_sec_query)

Event-Specific Information
```
char    *component_name
ulong_int     acl_type
```

The **rdacl_get_referral()** operation obtains a referral to an ACL update site. This function is used when the current ACL site yields a **sec_acl_site_readonly** error. Some replication managers will require all updates for a given object to be directed to a given replica. Clients of the generic ACL interface may know they are dealing with an object that is replicated in this way. This function allows them to recover from this problem and rebind to the proper update site. The client is required to have execute access on the parent of the object named by **component_name**.

Event Type (Event Number, Event Classes)
ACL_GetReferral (267 (0x10B), dce_sec_control, dce_sec_query)

Event-Specific Information
```
char    *component_name
uuid_t          manager_type
ulong_int      acl_type
```

Privilege Server Interface (rpriv) Operations: The **rpriv_get_ptgt()** operation returns a privilege certificate to the Ticket-granting service. The caller supplies the group set, and the Privilege Server seals the group set in the authorization portion of a privilege Ticket-granting ticket, after first rejecting any groups that are not legitimately part of the caller credentials. A group will be rejected if the caller is not a member of the group, or the group is not allowed on project lists (the **projlist_ok** flag is not set).

There is no access control on this interface other than what was within the Kerberos Ticket-granting mechanism itself; that is, the TGS request verification. This call may result in growth of potential access set. Note that this is a MVS/ESA OpenEdition DCE Release 1 routine.

Event Type (Event Number, Event Classes)
PRIV_GetPtgt (268 (0x10C), dce_sec_authent, dce_sec_control)

Event-Specific Information
```
char   *client_location
uuid_t     principal
uuid_t     group
short_int      num_groups  /* Number of local groups in PAC */
uuid_t     groups [] /* num_groups local groups in PAC */
```

Registry Server Account Interface (rs_acct) Operations: The **rs_acct_add()** operation adds an account with a specified login name. The caller needs to have **m**, **a**, and **u** (**mgmt_info**, **auth_info**, and **user_info**) permissions on the principal of the account that is to be added. The constituent principal, group, and organization (PGO) items for an account must be added before the account can be created. Also, the principal must have been added as a member of the specified group and organization.

Event Type (Event Number, Event Classes)
 ACCT_Add (269 (0x10D), dce_sec_control, dce_sec_modify)

Event-Specific Information

```
char      *account_name
long_int   key_parts
```

The **rs_acct_delete()** operation deletes an account with a specified login name. The caller needs to have **m**, **a**, and **u** (**mgmt_info**, **auth_info**, and **user_info**) permissions on the principal of the account that is to be deleted.

Event Type (Event Number, Event Classes)
 ACCT_Delete (270 (0x10E), dce_sec_control, dce_sec_modify)

Event-Specific Information

```
char      *account_name
```

The **rs_acct_rename()** operation changes the account login name. The caller has to have the **m** (**mgmt_info**) permission on the account's principal to be renamed (**old_login_name.pname**).

Event Type (Event Number, Event Classes)
 ACCT_Rename (271 (0x10F), dce_sec_control, dce_sec_modify)

Event-Specific Information

```
char      *old_account_name
char      *new_account_name
```

The **rs_acct_lookup()** operation returns data for a specified account. The caller must have the **r** (**read**) permission according to the ACL of the account's principal in order to be viewed.

Event Type (Event Number, Event Classes)
 ACCT_Lookup (272 (0x110), dce_sec_control, dce_sec_query)

Event-Specific Information

```
char      *account_name
```

The **rs_acct_replace()** operation replaces both the user and administrative information in the account record specified by the input login name. The administrative information contains limitations on the account's use and privileges. The user information contains such information as the account home directory and default shell. The administrative information can only be modified by a caller with the **a** (**auth_info**) privilege for the account's principal. The user information can be modified by a caller with the **u** (**user_info**) privileges for the account's principal.

Event Type (Event Number, Event Classes)
 ACCT_Replace (273 (0x111), dce_sec_control, dce_sec_modify)

Event-Specific Information

```
char      *account_name
long_int   key_parts
```

The **rs_acct_get_projlist()** operation returns members of the project list for the specified account. This operation requires the caller to have the **r** (**read**) permission on the account principal for which the project list data is to be returned.

Event Type (Event Number, Event Classes)
  ACCT_GetProjlist (274 (0x112), dce_sec_control, dce_sec_query)

Event-Specific Information

```
char      *account_name
long_int   key_parts
```

Registry Miscellaneous Operation Interface (rs_misc) Operations: The **rs_login_get_info()** operation returns login information for the specified account. This information is extracted from the account's entry in the registry database. This operation requires the caller to have the **r** (**read**) permission on the account's principal from which the data is to be returned.

Event Type (Event Number, Event Classes)
  LOGIN_GetInfo (275 (0x113), dce_sec_control, dce_sec_query)

Event-Specific Information

```
char      *principal_name
```

Registry PGO Interface (rs_pgo) Operations: The **rs_pgo_add()** operation adds a PGO item to the registry database. This operation requires the caller to have the **i** (**insert**) permission on the parent directory in which the PGO item is to be created.

Event Type (Event Number, Event Classes)
  PGO_Add (276 (0x114), dce_sec_control, dce_sec_modify)

Event-Specific Information

```
long_int   name_domain
char      *name
```

The **rs_pgo_delete()** operation deletes a PGO item from registry database. Any account depending on the deleted PGO item is also deleted. The deletion operation requires the caller to have the **d** (delete) permission on the parent directory that contains the PGO item to be deleted and the **D** (**Delete_object**) permission on the PGO item itself.

Event Type (Event Number, Event Classes)
  PGO_Delete (277 (0x115), dce_sec_control, dce_sec_modify)

Event-Specific Information

```
long_int   name_domain
char      *name
```

The **rs_pgo_replace()** operation replaces the data associated with a PGO item in the registry database. The caller needs to have the **m** (**mgmt_info**) permission on the PGO item, if **quota**, **flags**, or **unix_num** is being set. (Only a cell principal's **unix_num** is modifiable.) The caller needs to have the **f** (fullname) permission to modify the fullname of the PGO item.

Event Type (Event Number, Event Classes)
  PGO_Replace (278 (0x116), dce_sec_control, dce_sec_modify)

Event-Specific Information

```
long_int   name_domain
char      *name
```

The **rs_pgo_rename()** operation renames a PGO item in the registry database. The caller needs to have the **n** (**name**) permission on the old name of the PGO item, if performing a rename within a directory. In order to move a PGO item between directories, the caller needs to have the **n** (**name**) permission on the old name of the PGO item as well as the **d** (**delete**) permission on the old parent directory and the **i** (**insert**) permission on the new parent directory in which the PGO item is being added under the new name.

Event Type (Event Classes)
: **PGO_Rename (279 (0x117), dce_sec_control, dce_sec_modify)**

Event-Specific Information
```
long_int    name_domain
char  *old_name
char  *new_name
```

The **rs_pgo_get()** operation returns the name and data for a PGO item. The desired item is identified by a query key, which can be a **name**, a **uuid**, a **unix_num**, or a **sequential-search** flag. The caller needs to have the **r** (**read**) permission on the PGO item to be viewed.

Event Type (Event Number, Event Classes)
: **PGO_Get (280 (0x118), dce_sec_control, dce_sec_query)**

Event-Specific Information
```
long_int    name_domain
ulong_int    query_tag
ulong_int    query_value
```

The **rs_pgo_key_transfer()** operation performs a specified key transfer between the **uuid**, **unix_num**, and **name** of a PGO item. The caller needs to have some permission on the PGO item for **id->name** and **unix_num->name** transfers.

Event Type (Event Number, Event Classes)
: **PGO_KeyTransfer (281 (0x119), dce_sec_control)**

Event-Specific Information
```
long_int    name_domain
ulong_int    query_tag
ulong_int    query_value
ulong_int    result_type
```

The **rs_pgo_add_member()** operation adds a member to a group or an organization. The caller must have the **M** (**Member_list**) permission on the group or organization. Additionally, if this call is for adding a group member, the caller must have the **g** (**groups**) permission on the principal to be added.

Event Type (Event Number, Event Classes)
: **PGO_AddMember (282 (0x11A), dce_sec_control, dce_sec_modify)**

Event-Specific Information
```
long_int    name_domain
char    *person_name
char    *group/organization
```

The **rs_pgo_delete_member()** operation deletes a principal from a group or an organization in the registry database. The caller must have the **M** (**Member_list**) permission on the group or organization. Note that the caller does not need to have the **g** (**groups**) permission when deleting the principal from a group.

Event Type (Event Number, Event Classes)
: **PGO_DeleteMember (283 (0x11B), dce_sec_control, dce_sec_modify)**

Event-Specific Information
```
long_int    name_domain
char    *person_name
char    *group/organization
```

The **rs_pgo_is_member()** operation tests whether a specified principal is a member of a specified group or organization. The caller must have **t** (**test**) permission on the group or organization.

Event Type (Event Number, Event Classes)
PGO_IsMember (284 (0x11C), dce_sec_control, dce_sec_query)

Event-Specific Information
```
long_int    name_domain
char    *person_name
char    *group/organization
```

The **rs_pgo_get_members()** operation, if the specified domain is group or organization, lists the members of a specified group or organization. If the domain is principal, list the groups in which the principal is a member. The caller must have the **r** (**read**) permission on the principal, group, or organization.

Event Type (Event Number, Event Classes)
PGO_GetMembers (285 (0x11D), dce_sec_control, dce_sec_query)

Event-Specific Information
```
long_int    name_domain
char    *group/organization
```

Registry Policy Interface (rs_policy) Operations: The **rs_properties_get_info()** operation returns a list of registry properties. The caller must have the **r** (**read**) permission on the policy object from which the property information is to be returned.

Event Type (Event Number, Event Classes)
PROP_GetInfo (286 (0x11E), dce_sec_control, dce_sec_query)

Event-Specific Information
None

The **rs_properties_set_info()** operation sets the registry properties. The caller must have the **m** (**mgmt_info**) permission on the policy object for which the property information is to be set.

Event Type (Event Number, Event Classes)
PROP_SetInfo (287 (0x11F), dce_sec_control, dce_sec_modify)

Event-Specific Information
None

The **rs_policy_get_info()** operation returns the policy for a specified organization or the registry (if no organization name is specified). The caller must have the **r** (**read**) permission on the policy object or organization item from which the data is to be returned. Note that the **rs_policy_get_effective()** operation uses the same audit event (**POLICY_GetInfo**) as the **rs_policy_get_info()** operation.

Event Type (Event Number, Event Classes)
POLICY_GetInfo (288 (0x120), dce_sec_control, dce_sec_query)

Event-Specific Information
```
char   *organization
```

The **rs_policy_set_info()** operation sets the policy for a specified organization or the registry (if no organization name is specified). The caller must have the **m** (**mgmt_info**) permission on the policy object or organization item for which the data is to be set.

Event Type (Event Number, Event Classes)
POLICY_SetInfo (289 (0x121), dce_sec_control, dce_sec_modify)

Event-Specific Information
```
char   *organization
```

The **rs_auth_policy_get_info()** operation returns the authentication policy for a specified account or the registry (if no account is specified). The caller must have the **r** (**read**) permission on the policy object or account's principal from which the data is to be returned.

Event Type (Event Number, Event Classes)
    AUTHPOLICY_GetInfo (290 (0x122), dce_sec_control, dce_sec_query)

Event-Specific Information
```
char  *account_name
```

The **rs_auth_policy_get_effective()** operation returns the effective authentication policy for an account. If no account is specified, the authentication policy for the registry is returned. The caller must have the **r** (**read**) permission on the policy object of the registry. If an account is specified, the caller must also have **r** (**read**) permission on the account's principal.

Event Type (Event Number, Event Classes)
    No new event is defined for this operation. **AUTHPOLICY_GetInfo** is used here.

The **rs_auth_policy_set_info()** operation sets the authentication policy for an account or the registry (if no account is specified). The caller must have the **a** (**auth_info**) permission on the account's principal or policy object of the registry.

Event Type (Event Number, Event Classes)
    AUTHPOLICY_SetInfo (291 (0x123), dce_sec_control, dce_sec_modify)

Event-Specific Information
```
char  *account_name
```

Registry Administration Interface Operations: The **rs_rep_admin_stop()** operation directs the registry server to stop servicing remote procedure calls. The caller must have **A** (**Admin**) permission on the registry policy object.

Event Type (Event Number, Event Classes)
    REPADMIN_Stop (292 (0x124), dce_sec_control, dce_sec_server)

Event-Specific Information
    None

The **rs_rep_admin_maint()** operation directs the registry server into (checkpoint the database, close files, and so on) or out of maintenance state. The caller must have **A** (**Admin**) permission on the registry policy object.

Event Type (Event Number, Event Classes)
    REPADMIN_Maint (293 (0x125), dce_sec_control, dce_sec_server)

Event-Specific Information
```
boolean  in_maintenance_mode
```

The **rs_rep_admin_mkey()** operation directs the registry to change its master key and re-encrypt account keys using the new master key. The caller must have **A** (**Admin**) permission on the registry policy object.

Event Type (Event Number, Event Classes)
    REPADMIN_Mkey (294 (0x126), dce_sec_control, dce_sec_server)

Event-Specific Information
    None

The **rs_rep_admin_destroy()** operation directs the registry server replica to destroy its database and exit. The caller must have **A** (**Admin**) permission on the registry policy object.

Event Type (Event Classes)
REPADMIN_Destroy (295 (0x127), dce_sec_control, dce_sec_server)

Event-Specific Information
None

The **rs_rep_admin_init_replica()** operation directs the registry server to (re-)initialize the slave identified by *rep_id*. This is a master server only operation. The caller must have **A** (**Admin**) permission on the registry policy object.

Event Type (Event Classes)
REPADMIN_Init (296 (0x128), dce_sec_control, dce_sec_server)

Event-Specific Information
```
char    *replica_identifier
```

The **rs_rep_admin_set_sev_rev()** operation sets the cell software revision level. The caller must have **A** (**Admin**) permission on the registry policy object.

Event Type (Event Classes)
REPADMIN_SetSwRev (320 (0x140), dce_sec_control, dce_sec_server)

Event-Specific Information
```
ulong_int  software_revision_level
```

Identifier Mapping Interface (secidmap) Operations: The **rsec_id_parse_name()** operation translates a global name into principal and cell names and UUIDs. If the principal's UUID is requested, the caller must have at least one permission of any kind on the principal item.

Event Type (Event Number, Event Classes)
SECID_ParseName (297 (0x129), dce_sec_control)

Event-Specific Information
```
char    global_name
```

The **rsec_id_gen_name()** operation generates a global name from cell and principal UUIDs. The caller must have at least one permission of any kind on the specified principal.

Event Type (Event Number, Event Classes)
SECID_GenName (298 (0x12A), dce_sec_control)

Event-Specific Information
```
char    global_name
```

Registry Server Attributes Manipulation Interface (rs_attr) Operations: The **rs_attr_update()** operation updates (writes/creates) an attribute. The caller must have, for each attribute defined in **attr_keys**, the **query_permset** permission on the registry object specified.

Event Type (Event Classes)
ERA_Update (299 (0x12B), dce_sec_control, dce_sec_modify)

Event-Specific Information
```
char  *component_name
ulong_int  attribute_count
uuid  attribute_uuid
```

The **rs_attr_delete()** operation deletes a specified attribute(s). The caller must have **delete_permset** permission for each attribute specified.

Event Type (Event Classes)
  ERA_Delete (300 (0x12C), dce_sec_control, dce_sec_modify)

Event-Specific Information
```
char  *component_name
ulong_int  attribute_count
uuid  attribute_uuid
```

The **rs_attr_lookup_by_id()** operation performs a lookup of the attributes by attribute type ID. If the number of query attribute keys is 0, this operation will return all attributes that the caller is authorized to use. The caller must have, for each attribute specified, the **query_permset** permission on the registry object specified.

Event Type (Event Classes)
  ERA_LookupById (302 (0x12E), dce_sec_control)

Event-Specific Information
```
char  *component_name
ulong_int  attribute_count
uuid  attribute_uuid
```

The **rs_attr_lookup_no_expand()** operation performs a lookup of the attributes by attribute type ID without expanding attribute sets to their constituent member attributes. If the number of query attribute keys is 0, this operation will return all attributes that the caller is authorized to use. The caller must have, for each attribute specified, the **query_permset** permission on the registry object specified.

Event Type (Event Classes)
  ERA_LookupNoExpand (303 (0x12F), dce_sec_control)

Event-Specific Information
```
char  *component_name
ulong_int  attribute_count
uuid  attribute_uuid
```

The **rs_attr_lookup_by_name()** operation performs a lookup of an attribute by name. The caller must have, for the attribute specified, **query_permset** permission on the registry object specified.

Event Type (Event Classes)
  ERA_LookupByName (304 (0x130), dce_sec_control)

Event-Specific Information
```
char    *component_name
char    *attribute_name
```

Registry Server Attributes Schema Manipulation Interface (rs_attr_schema)

Operations: The **rs_attr_schema_create_entry()** operation creates a new schema entry. The caller must be authorized to add entries to the specified schema.

Event Type (Event Classes)
  ERA_SchemaCreate (305 (0x131), dce_sec_control, dce_sec_modify)

Event-Specific Information
```
char    *schema_name
char    *attribute_name
uuid    attribute_uuid
```

The **rs_attr_schema_delete_entry()** operation deletes a schema entry. The caller must be authorized to delete schema entries.

Event Type (Event Classes)

ERA_SchemaDelete (306 (0x132), dce_sec_control, dce_sec_modify)

Event-Specific Information

```
char    *schema_name
uuid    attribute_uuid
```

The **rs_attr_schema_update_entry()** operation updates the modifiable fields of a schema entry. The caller needs to have **m (mgmt_info)** permissions on the schema entry that is to be modified.

Event Type (Event Classes)

ERA_SchemaUpdate (307 (0x133), dce_sec_control, dce_sec_modify)

Event-Specific Information

```
char    *schema_name
uuid    attribute_uuid
```

The **rs_attr_schema_lookup_by_id()** operation retrieves the schema entry identified by the attribute type **uuid**. The caller must have **r (read)** permissions on the schema entry specified.

Event Type (Event Classes)

ERA_SchemaLookupById (308 (0x134), dce_sec_control)

Event-Specific Information

```
char    *schema_name
uuid    attribute_uuid
```

The **rs_attr_schema_lookup_by_name()** operation retrieves the schema entry identified by the attribute name. The caller must have **r (read)** permissions on the schema entry specified.

Event Type (Event Classes)

ERA_SchemaLookupByName (309 (0x135), dce_sec_control)

Event-Specific Information

```
char    *schema_name
uuid    attribute_uuid
```

OS/390 DCE Release 1 Privilege Server Manager Interface (rpriv_v1_1)

Operations: The **rpriv_get_eptgt()** operation constructs and returns an extended privilege certificate to the ticket_granting service. The caller supplies the extended privilege attributes in the form of an encoded Extended Privilege Attribute Certificate (EPAC). The procedure by which the requested privilege attributes are verified depends on how the call is authenticated and whether the request is “local” (that is, is a request from a client in this Privilege Server's cell) or is “intercell” (that is, is from a foreign privilege service).

If the request is local, then the ticket to the Privilege Server is based on a Kerberos V5 TGT and the **requested_privs** consists of a single encoded EPAC. The Privilege Server decodes the **requested_privs** and verifies that the requested privileges are valid by performing the necessary database queries.

If the request is foreign, then the ticket to the privilege service is based on a DCE EPTGT and the Privilege Server retrieves the EPAC seal from the DCE authorization data contained in the ticket, and uses it to verify that the requested privileges are valid.

Event Type (Event Classes)

PRIV_GetEptgt (310 (0x136), dce_sec_control, dce_sec_authent)

Event-Specific Information

```
char     *client_location
uuid     principal
uuid     group
short_int     number_local_groups
uuid     local_group
number_certificates

For each EPAC:
  uuid    realm
  uuid    principal
  uuid    group
  ushort_int     number_group_records
  uuid    group
```

The **rpriv_become_delegate()** operation permits an intermediate server to become a delegate for its caller. The caller supplies extended privilege attributes in the form of an encoded Extended Privilege Attribute Certificate (EPAC). The Privilege Server verifies that the delegation token for this EPAC chain is correct and then creates a new chain from the existing one with the intermediary's EPAC as a new delegate.

Event Type (Event Classes)
: **PRIV_BecomeDelegate (312 (0x138), dce_sec_control, dce_sec_authent)**

Event-Specific Information

```
char     *client_location
uuid     principal
uuid     group
short_int     number_local_groups
uuid     local_group
number_certificates

For each EPAC:
  uuid    realm
  uuid    principal
  uuid    group
  ushort_int     number_group_records
  uuid    group
```

The **rpriv_become_impersonator()** operation permits an intermediate server to become an impersonator for its caller. The caller supplies extended privilege attributes in the form of an encoded Extended Privilege Attribute Certificate (EPAC). The Privilege Server verifies that the delegation token for the initator's EPAC is correct and also that the intermediary is allowed to impersonate the initiator.

Event Type (Event Classes)
: **PRIV_BecomeImpersonator (313 (0x139), dce_sec_control, dce_sec_authent)**

Event-Specific Information

```
char     *client_location
uuid     principal
uuid     group
short_int     number_local_groups
uuid     local_group
number_certificates

For each EPAC:
  uuid    realm
  uuid    principal
  uuid    group
```

```
ushort_int     number_group_records
uuid    group
```

Related Information

Commands:

- **dcecp**

Files:

dce_sec_authent dce_sec_control dce_sec_modify dce_sec_query dce_sec_server

Glossary

This glossary defines new OS/390 DCE terms and abbreviations used in the OS/390 DCE library of publications. If you do not find the term you are looking for, refer to the index or to the *IBM Dictionary of Computing*, SC20-1699.

This glossary includes terms and definitions from:

- *IBM Dictionary of Computing*, SC20-1699.
- *Information Technology—Portable Operating System Interface (POSIX),* from the POSIX series of standards for applications and user interfaces to open systems, copyrighted by the Institute of Electrical and Electronics Engineers (IEEE).
- *American National Standard Dictionary for Information Systems,* ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI).
- *Information Technology Vocabulary,* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1.SC1).
- *CCITT Sixth Plenary Assembly Orange Book, Terms and Definitions* and working documents published by the International Telecommunication Union, Geneva, 1978.
- Open Software Foundation (OSF).

The following abbreviations indicate terms that are related to a particular DCE service:

CDS	Cell Directory Service
DTS	Distributed Time Service
GDS	Global Directory Service
RPC	Remote Procedure Call
Security	Security Service
Threads	Threads Service
XDS	X/OPEN Directory Service
XOM	X/OPEN Object Management

A

absolute time. A point on a time scale.

abstract syntax notation one (ASN.1). A data representation scheme that enables complicated types to be defined and enables values of these types to be specified.

access control list (ACL). (1) GDS: Specifies the users with their access rights to an object. (2) Security: Data that controls access to a protected object. An ACL specifies the privilege attributes needed to access the object and the permissions that may be granted, to the protected object, to principals that possess such privilege attributes.

access right. Synonym for *permission*.

accessible. Pertaining to an object whose client possesses a valid designator or handle.

account. Data in the Registry database that allows a principal to log in. An account is a registry object that relates to a principal.

ACL. Access control list.

adapter. Synonym for *attachment facility*.

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

address family. A set of related communications protocols that use a common addressing mechanism to identify end-points; for example, the U.S. Department of Defense Internet Protocols. Synonymous with *protocol family*.

aename. An option used in Workload Balancing commands. A string, up to 18 bytes in length, referring to Application Environment.

alias. Synonym for *alias name*.

alias name. (1) GDS: A name for a directory object that consists of one or more alias entries in the directory information tree (DIT). (2) Security: An optional alternate for a principal's primary name. Synonymous with *alias*. The alias shares the same UUID with the primary name.

aliasing. RPC: Pertaining to the pointing of two pointers of the same operation at the same storage.

APF. Authorized program facility.

API. Application program interface.

application program interface (API). A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

Application Support Server. Refers to the server for OS/390 DCE Application Support. The Application Support server allows a client program to access CICS® or IMS™.

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

architecture. (1) The organizational structure of a computer system, including the interrelationships among its hardware and software. (2) The logical structure and operating principles of a computer network. The operating principles of a network include those of services, functions, and protocols.

ASN.1. Abstract syntax notation one.

association (connection-oriented). A connection between a client and a server.

attachment facility. Application Support Server: Refers to the CICS adapter and the IMS adapter. Synonymous with *adapter*.

attribute. (1) RPC: An Interface Definition Language (IDL) or attribute configuration file (ACF) that conveys information about an interface, type, field, parameter, or operation. (2) DTS: A qualifier used with DTS commands. DTS has four attribute categories: characteristics, counters, identifiers, and status. (3) XDS: Information of a particular type concerning an object and appearing in an entry that describes the object in the directory information base (DIB). It denotes the attribute's type and a sequence of one or more attribute values, each accompanied by an integer denoting the value's syntax.

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

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

attribute type. (1) XDS: The component of an attribute that indicates the type of information given by that attribute. Because it is an object identifier, it is unique among other attribute types. (2) XOM: Any of various categories into which the client dynamically groups values on the basis of their semantics. It is an integer unique only within the package.

attribute value. XDS, XOM: A particular instance of the type of information indicated by an attribute type.

attribute value assertion (AVA). GDS: An attribute type and attribute value pair. A relative distinguished name is comprised of one or more AVAs.

authentication. In computer security, a method used to verify the identity of a principal.

authentication level. Synonym for *protection level*.

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

Authentication Service. One of three services provided by the Security Service: it verifies principals according to a specified authentication protocol. The other Security services are the Privilege Service and the Registry Service.

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

authorization service. RPC: An implementation of an authorization protocol.

AVA. Attribute value assertion.

B

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

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

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

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

binding handle. RPC: A reference to a binding. See *binding information*.

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

See *binding*. See also *endpoint*, *network address*, *RPC protocol*, *RPC protocol sequence*, and *transfer syntax*.

Boolean. Boolean algebra.

The type of an expression with two binary values, “true” and “false”. Also, a variable of Boolean type or a function with Boolean arguments or result. The most common Boolean functions are AND, OR, and NOT.

In DCE Workload Balancing, when a server is workload balanced, its activate bit takes on a value of 1 (TRUE); when not balanced its activate bit takes on a value of 0 (FALSE).

broadcast. A notification sent to all members within an arbitrary grouping such as nodes in a network or threads in a process. See also *signal*.

C

cache. (1) CDS: The information that a CDS clerk stores locally to optimize name lookups. The cache contains attribute values resulting from previous lookups, as well as information about other clearinghouses and namespaces. (2) Security: Contains the credentials of a principal after the DCE login. (3) GDS: See *DUA cache*.

call thread. RPC: A thread created by an RPC server’s runtime to run remote procedures. When engaged by a remote procedure call, a call thread temporarily forms part of the RPC thread of the call. See *application thread* and *RPC thread*.

cancel. (1) Threads: A mechanism by which a thread informs either itself or another thread to stop the thread as soon as possible. If a cancel arrives during an important operation, the canceled thread may continue until it can end the thread in a controlled manner.
(2) RPC: A mechanism by which a client thread notifies a server thread (the canceled thread) to end the thread as soon as possible. See also *thread*.

CCITT. Consultative Committee on International Telegraphy and Telephone

CDS. Cell Directory Service.

CDS clerk. The software that provides an interface between client applications and CDS servers.

CDS control program (CDSCP). A command interface that CDS administrators use to control CDS servers and clerks and manage the namespace and its contents. See also *manager*.

CDSCP. CDS control program.

cell. The basic unit of operation in the distributed computing environment. A cell is a group of users, systems, and resources that are grouped around a common purpose and that share common DCE services.

Cell Directory Service (CDS). A DCE component. A distributed replicated database service that stores names and attributes of resources located in a cell. CDS manages a database of information about the resources in a group of machines called a DCE cell.

cell-relative name. Synonym for *local name*.

chaining. GDS, XDS: A mode of interaction optionally used by a directory system agent (DSA) that cannot perform an operation itself. The DSA chains by calling the operation in another DSA and then relaying the outcome to the original requester.

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

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

child process. A process, created by a parent process, that shares the resources of the parent process to carry out a request. Contrast with *parent process*. See also *fork*.

CICS. Customer Information Control System.

class. A category into which objects are placed on the basis of their purpose and internal structure.

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

clearinghouse. CDS: A collection of directory replicas on one CDS server. A clearinghouse takes the form of a database file. It can exist only on a CDS server node; it cannot exist on a node running only CDS clerk software. Usually only one clearinghouse exists on a server node.

clearinghouse object entry. CDS: A special class of object entry that describes a clearinghouse. The clearinghouse object entry is a pointer to the network address of an actual clearinghouse. This pointer enables CDS to find a clearinghouse and use and manage its contents. A clearinghouse changes and manages its own object entry when necessary. The clearinghouse object entry has the same name as the clearinghouse it describes.

clerk. (1) DTS: A software component that synchronizes the clock for its client system by requesting time values from servers, calculating a new time from the values, and supplying the computed time to client applications. (2) CDS: A software component that receives CDS requests from a client application, ascertains an appropriate CDS server to process the requests, and returns the results of the requests to the client application.

client. A computer or process that accesses the data, services, or resources of another computer or process on the network. Contrast with *server*.

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

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

client context. RPC: The state within an RPC server generated by a set of remote procedures and maintained across a series of calls for a particular client. See *context handle*. See also *manager*.

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

client/server model. A form of computing where one system, the client, requests something, and another system, the server, responds.

clock. The combined hardware interrupt timer and software register that maintains the system time.

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

code page. (1) A table showing codes assigned to character sets. (2) An assignment of graphic characters and control function meanings to all code points. (3) Arrays of code points representing characters that establish numeric order of characters. [OSF] (4) A particular assignment of hexadecimal identifiers to graphic elements. (5) Synonymous with code set. (6) See also *code point, extended character*.

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

compatible server. RPC: A server that offers the requested RPC interface and RPC object and that is accessible over a valid combination of network and transport protocols. It is supported by both the client and server RPC run times.

computed time. DTS: The resulting time after a DTS clock synchronization. The time value that the clerk or server process computes according to the values it receives from several servers.

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

Telephone (CCITT)

Consultative Committee on International Telegraphy and. A United Nations Specialized Standards group whose membership includes common carriers concerned with devising and proposing recommendations for international telecommunications representing alphabets, graphics, control information, and other fundamental information interchange issues.

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

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

control task. The parent process of the DCE daemons in the DCEKERN address space. All requests to start or stop DCE daemons are handled by the Control Task.

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

conversation key. Synonym for *session key*.

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

courier. DTS: A local server that requests a time value from a randomly selected global server. The time value returned is used for synchronization.

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

credentials. Security: A general term for privilege attribute data that has been certified by a trusted privilege certification authority.

cross-linking information. In order for OS/390 DCE to provide RACF-DCE interoperability and single sign-on to DCE, DCE provides utilities (see **mvsexpt** and **mvsimpt**) to incorporate into RACF the information that associates an OS/390-RACF user ID with a DCE principal's identifying information and the DCE principal's UUID with the corresponding OS/390-RACF user ID. The information is placed in a RACF DCE segment and the RACF general resource class, DCEUUIDS. This is called **cross-linking information** and is what allows interoperability and single sign-on to work. See also *interoperability* and *single sign-on*.

CTS. Creation timestamp.

Customer Information Control System (CICS). An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining databases.

D

daemon. (1) A long-lived process that runs unattended to perform continuous or periodic system-wide functions such as network control Some daemons are triggered automatically to perform their task; others operate periodically. An example is the **cron** daemon, which periodically performs the tasks listed in the **crontab** file. Many standard dictionaries accept the spelling *demon*. (2) A DCE server process.

daemon configuration file. A file containing information on which daemons are configured on the host, which environment variables to set, the parameters to pass to the process, minimum restart interval, and the time-out period.

Data Encryption Standard (DES). The National Institute of Standards and Technology (NIST) Data Encryption Standard, adopted by the U.S. government as Federal Information Processing Standard (FIPS) Publication 46, which allows only hardware implementations of the data encryption algorithm.

datagram. RPC: A network data packet that is independent of all other packets and does not guarantee delivery or sequentiality.

datagram protocol. RPC: A datagram-based transport protocol, such as User Datagram Protocol (UDP), that runs over a connectionless transport protocol.

DCE. Distributed Computing Environment.

DCECONF. program used to configure and start the DCE daemons.

DCEKERN. The address space that contains the DCE daemons.

decrypt. Security: To decipher data.

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

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

DES. Data Encryption Standard.

DFS. Distributed File Service.

DIB. Directory information base.

directory. (1) A logical unit for storing entries under one name (the directory name) in a CDS namespace. Each physical instance of a directory is called a replica. (2) A collection of open systems that cooperates to hold a logical database of information about a set of objects in the real world.

directory ID. Directory identifier.

directory information base (DIB). GDS: The complete set of information to which the directory provides access, which includes all of the pieces of information that can be read or manipulated using the operations of the directory.

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

directory schema. GDS: The set of rules and constraints concerning directory information tree (DIT) structure, object class definitions, attribute types, and syntaxes that characterize the directory information base (DIB).

Directory Service. A DCE component. The Directory Service is a central repository for information about resources in a distributed system. See *Cell Directory Service* and *Global Directory Service*.

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

directory system agent (DSA). GDS: An open systems interconnection (OSI) application process that is part of the directory.

directory user agent (DUA). GDS: An open systems interconnection (OSI) application process that represents a user accessing the directory.

distinguished name (DN). GDS: One of the names of an object, formed from the sequence of RDNs of its object entry and each of its superior entries.

distinguished value. GDS: An entry's attribute value that has been designated to appear in the RDN of the entry.

distributed computing. A type of computing that allows computers with different hardware and software to be combined on a network, to function as a single computer, and to share the task of processing application programs.

Distributed Computing Environment (DCE). A comprehensive, integrated set of services that supports the development, use, and maintenance of distributed applications. DCE is independent of the operating system and network; it provides interoperability and portability across heterogeneous platforms.

Distributed File Service (DFS). A DCE component. DFS joins the local file systems of several file server machines making the files equally available to all DFS client machines. DFS allows users to access and share files stored on a file server anywhere in the network, without having to consider the physical location of the file. Files are part of a single, global namespace, so that a user can be found anywhere in the network by means of the same name.

distributed service. A DCE service that is used mainly by administrators to manage a distributed environment. These services include DTS, Security, and Directory.

Distributed Time Service (DTS). A DCE component. It provides a way to synchronize the times on different hosts in a distributed system.

DIT. Directory information tree.

DN. Distinguished name.

DNS. Domain Name System.

Domain Name System (DNS). A hierarchical scheme for giving meaningful names to hosts in a TCP/IP network.

domain name. A unique network name that is associated with a network's unique address.

drift. DTS: The change in a clock's error rate over a specified period of time.

DSA. Directory system agent.

DTS. Distributed Time Service.

DTS entity. DTS: The server or clerk software on a system.

DUA. Directory user agent.

DUA cache. GDS: The part of the DUA that stores information to optimize name lookups. Each cache contains copies of recently accessed object entries as well as information about DSAs in the directory.

dynamic endpoint. RPC: An endpoint that is generated by the RPC runtime for an RPC server when the server registers its protocol sequences. It expires when the server stops running. See *endpoint* and *well-known endpoint*.

E

effective permissions. Security: The permissions granted to a principal as a result of a masking operation.

element. RPC: Any of the bits of a bit string, the octets of an octet string, or the octets by means of which the characters of a character string are represented.

encrypt. To systematically encode data so that it cannot be read without knowing the coding key.

endian. An attribute of data representation that reflects how certain multi-octet data is stored in memory. See *big endian* and *little endian*.

endpoint. RPC: An address of a specific server instance on a host.

endpoint map. RPC: A database local to a node where local RPC servers register binding information associated with their interface identifiers and object identifiers. The endpoint map is maintained by the endpoint map service of the DCE daemon.

endpoint map service. RPC: A service that maintains a system's endpoint map for local RPC servers. When an RPC client makes a remote procedure call using a partially bound binding handle, the endpoint map service looks up the endpoint of a compatible local server. See *endpoint map*.

entity. (1) CDS: Any manageable element through the CDS namespace. Manageable elements include directories, object entries, servers, replicas, and clerks. The CDS control program (CDSCP) commands are based on directives targeted for specific entities. (2) DTS: See *DTS entity*.

entity type. DTS: An identifier of an entity that determines whether it is a server or a clerk.

entry. GDS, XDS: The part of the DIB that contains information relating to a single directory object. Each entry consists of directory attributes.

ENV. environment variable

environment variable (ENV). A variable included in the current software environment that is available to any called program that requests it.

epoch number. DTS: An attribute that a server appends to the time values it sends to other servers. Servers use time values only from other servers with whom they share epoch numbers.

error tolerance. DTS: The amount of system clock inaccuracy to which the DCE Time Service responds by abruptly setting the system clock to the computed time, rather than gradually adjusting the clock.

exception. (1) An abnormal condition such as an I/O error encountered in processing a data set or a file. (2) One of five types of errors that can occur during a floating-point exception. These are valid operation, overflow, underflow, division by zero, and inexact results. [OSF] (3) Contrast with *interrupt, signal*.

executor thread. See *call thread*.

export. (1) RPC: To place the server binding information associated with an RPC interface or a list of object UUIDs or both into an entry in a name service database. (2) To provide access information for an RPC interface. Contrast with *unexport*.

F

fault. RPC: An exception condition, occurring on a server, that is transmitted to a client.

filter. An assertion about the presence or value of certain attributes of an entry to limit the scope of a search.

foreign cell. A cell other than the one to which the local machine belongs. A foreign cell and its binding information are stored in either GDS or the Domain Name System (DNS). The act of contacting a foreign cell is called intercell. Contrast with *local cell*.

fork. To create and start a child process. Forking is similar to creating an address space and attaching. It creates a copy of the parent process, including open file descriptors.

full name. CDS: The complete specification of a CDS name, including all parent directories in the path from the cell root to the entry being named.

fully bound binding handle. RPC: A server binding handle that contains a complete server address including an endpoint. Contrast with *partially bound binding handle*.

G

GDA. Global Directory Agent.

GDS. Global Directory Service.

Global Directory Agent (GDA). A DCE component that makes it possible for the local CDS to access names in foreign cells. The GDA provides a connection to foreign cells through either the GDS or the Domain Name System (DNS).

Global Directory Service (GDS). A DCE component. A distributed replicated directory service that provides a global namespace that connects the local DCE cells into one worldwide hierarchy. DCE users can look up a name outside a local cell with GDS.

global name. A name that is universally meaningful and usable from anywhere in the DCE naming environment. The prefix /... indicates that a name is global.

global server. DTS: A server that provides its clock value to courier servers on other cells, or to DTS entities that have failed to obtain the specified number of servers locally.

global set. DTS: The group of global servers in a network.

group. (1) RPC: A name service entry that corresponds to one or more RPC servers that offer common RPC interfaces, RPC objects, or both. A group contains the names of the server entries, other groups, or both that are members of the group. See *NSI group attribute*. (2) Security: Data that associates a named set of principals that can be granted common access rights. See *subject identifier*.

group member. (1) RPC: A name service entry whose name occurs in the group. (2) Security: A principal whose name appears in a security group. See *group*.

H

handle. RPC: An opaque reference to information. See *binding handle*, *context handle*, *interface handle*, *name service handle*, and *thread handle*.

heterogeneous. Pertaining to a collection of dissimilar host computers such as those from different manufacturers. Contrast with *homogeneous*.

high convergence. CDS: A setting that controls the degree to which CDS attempts to keep all replicas of a directory consistent. High convergence means CDS makes one attempt to immediately propagate an update to all replicas. If that attempt fails (for example, if one of the replicas is unavailable), the software schedules a skulk for within one hour. Under usual circumstances, a skulk occurs at least once every twelve hours on a directory with high convergence. Setting a directory's **CDS_Convergence** attribute controls convergence. See *low convergence* and *medium convergence*.

home cell. Synonym for *local cell*.

homogeneous. Pertaining to a collection of similar host computers such as those of one model or one manufacturer. Contrast with *heterogeneous*.

host ID. Synonym for *network address*.

I

identity mapping. Application Support Server: A record in the Security Registry that contains the mapping between a client's DCE identity and an OS/390 user ID.

IDL. Interface Definition Language.

IDL compiler. RPC: A compiler that processes an RPC interface definition and an optional attribute configuration file (ACF) to generate client and server stubs, and header files. See *Interface Definition Language*.

import. (1) RPC: To obtain binding information from a name service database about a server that offers a given RPC interface by calling the RPC NSI import operation. (2) RPC: To incorporate constant, type, and import declarations from one RPC interface definition into another RPC interface definition by means of the IDL import statement.

IMS. Information Management System.

inaccuracy. DTS: The bounded uncertainty of a clock value as compared to a standard reference.

Information Management System (IMS). A database and data communication system capable of managing complex databases and networks in virtual storage.

interoperability. The capability to communicate, run programs, or transfer data among various functional units in a way that requires the user to have little or no knowledge of the unique characteristics of those units.

instance. XOM: An object in the category represented by a class.

integrity. RPC: A protection level that may be specified in secure RPC communications to ensure that data transferred between two principals has not been changed in transit.

interface. RPC: A shared boundary between two or more functional units, defined by functional characteristics, signal characteristics, or other characteristics, as appropriate. The concept includes the specification of the connection of two devices having different functions. See *RPC interface*.

interface definition. RPC: A description of an RPC interface written in the DCE Interface Definition Language (IDL). See *RPC interface*.

Interface Definition Language (IDL). A high-level declarative language that provides syntax for interface definitions.

interface handle. RPC: A reference in code to an interface specification. See *binding handle* and *interface specification*.

interface identifier. RPC: A string containing the interface Universal Unique Identifier (UUID) and major and minor version numbers of a given RPC interface. See *RPC interface*.

interface specification. RPC: An opaque data structure that is generated by the DCE IDL compiler from an interface definition. It contains identifying and descriptive information about an RPC interface. See *interface definition*, *interface handle*, and *RPC interface*.

interface UUID. RPC: The Universal Unique Identifier (UUID) generated for an RPC interface definition using the UUID generator. See *interface definition* and *RPC interface*.

International Organization for Standardization (ISO). An international body composed of the national standards organizations of 89 countries. ISO issues standards on a vast number of goods and services including networking software.

Internet address. The 32-bit address assigned to hosts in a TCP/IP network.

Internet Protocol (IP). In TCP/IP, a protocol that routes data from its source to its destination in an Internet environment. IP provides the interface from the higher level host-to-host protocols to the local network protocols. Addressing at this level is usually from host to host.

interval. DTS: The combination of a time value and the inaccuracy associated with it; the range of values represented by a combined time and inaccuracy notation. As an example, the interval 08:00.00I00:05:00 (eight o'clock, plus or minus five minutes) contains the time 07:57.00.

IP. Internet Protocol

ISO. International Organization for Standardization

J

junction. A specialized entry in the DCE namespace that contains binding information to enable communications between different DCE services.

K

Kerberos. The authentication protocol used to carry out DCE private key authentication. Kerberos was developed at the Massachusetts Institute of Technology.

key. A value used to encrypt and decrypt data.

key file. A file that contains encryption keys for noninteractive principals.

L

LAN. Local area network.

layer. In network architecture, a group of services, functions, and protocols that is complete from a conceptual point of view, that is one out of a set of hierarchically arranged groups, and that extends across all systems that conform to the network architecture.

leaf entry. A directory entry that has no subordinates. It can be an alias entry or an object entry.

leap seconds. An infrequent adjustment to coordinated universal time to account for the irregularity of the earth's rotation.

little endian. An attribute of data representation that reflects how multi-octet data is stored. In little endian representation, the lowest addressed octet of a multi-octet data item is the least significant. See *big endian*.

local. (1) Pertaining to a device directly connected to a system without the use of a communication line. (2) Pertaining to devices that have a direct, physical connection. Contrast with *remote*.

local application thread. RPC: An application thread that runs within the confines of one address space on a local system and passes control exclusively among local code segments. See *application thread*, *client application thread*, *RPC thread* and *server application thread*.

local area network (LAN). A network in which communication is limited to a moderate-sized geographical area (1 to 10 km) such as a single office building, warehouse, or campus, and which does not generally extend across public rights-of-way. A local network depends on a communication medium capable of moderate to high data rate (greater than 1Mbps), and normally operates with a consistently low error rate.

local cell. The cell to which the local machine belongs. Synonymous with *home cell*. Contrast with *foreign cell*.

local DSA. GDS: A directory service agent (DSA) that is resident on the same computer as the directory user agent (DUA).

local name. A name that is meaningful and usable only within the cell where an entry exists. The local name is a shortened form of a global name. Local names begin with the prefix /.: and do not contain a cell name. Synonymous with *cell-relative name*.

local server. DTS: A server that synchronizes with its peers and provides its clock value to other servers and clerks in the same network.

local set. DTS: A collection of the servers in a particular network.

login facility. A Security Service facility that enables a principal to establish its identity.

low convergence. A setting that controls the degree to which CDS attempts to keep all replicas of a directory consistent. Low convergence means CDS does not immediately propagate an update; it simply waits for the next skulk to distribute all updates that occurred since the last skulk. Skulks occur at least once every 24 hours on directories with low convergence. Low convergence helps conserve resources by avoiding update propagations between skulks. Setting a directory's **CDS_Convergence** attribute controls convergence. See *high convergence* and *medium convergence*.

M

manager. RPC: A set of remote procedures that implement the operations of an RPC interface and that can be dedicated to a given type of object. See also *object* and *RPC interface*.

mask. (1) A pattern of characters used to control the retention or deletion of portions of another pattern of characters (2) Security: Used to establish maximum permissions that can then be applied to individual ACL entries. (3) GDS: The administration screen interface menus.

master replica. CDS: The first instance of a specific directory in the namespace. After copies of the directory have been made, a different replica can be designated as the master, but only one master replica of a directory can exist at a time. CDS can create, update, and delete object entries and soft links in a master replica.

medium convergence. CDS: A setting that controls the degree to which CDS attempts to keep all replicas of a directory consistent. Medium convergence means CDS makes one attempt to immediately propagate an update to all replicas of the directory in which a change was made. If the attempt fails, the software lets the next scheduled skulk make the replicas consistent. Skulks occur at least once every 12 hours on a directory with medium convergence. When a namespace is created, the default setting on the root directory is medium. Setting a directory's **CDS_Convergence** attribute controls convergence. See *high convergence* and *low convergence*.

minimum restart interval. The minimum amount of time in seconds that a stopped DCE daemon must have been running before it can be restarted automatically.

MODIFY DCEKERN. MODIFY command used to start, stop, and display the status of DCE daemons.

mvsexpt. One of two (the other is **mvsimpt**) utilities used to automate much of the administrator's work in creating the cross-linking information for DCE-RACF interoperability. The **mvsexpt** utility creates the cross-linking information in the RACF database from information in the DCE registry. See also *cross-linking information*, *interoperability*, and *single sign-on*.

mvsimpt. One of two (the other is **mvsexpt**) utilities used to automate much of the administrator's work in creating the cross-linking information for DCE-RACF interoperability. The **mvsimpt** utility creates DCE principals from information obtained from the RACF database. See also *cross-linking information*, *interoperability*, and *single sign-on*.

N

name. GDS, CDS: A construct that singles out a particular (directory) object from all other objects. A name must be unambiguous (denote only one object); however, it need not be unique (be the only name that unambiguously denotes the object).

name service. A central repository of named resources in a distributed system. In DCE, this is the same as Directory Service.

name service handle. RPC: An opaque reference to the context used by the series of next operations called during a specific name service interface (NSI) search or inquiry.

name service interface (NSI). RPC: A part of the application program interface (API) of the RPC run time. NSI routines access a name service, such as CDS, for RPC applications.

namespace. CDS: A complete set of CDS names that one or more CDS servers look up, manage, and share. These names can include directories, object entries, and soft links.

NDR. Network Data Representation.

network. A collection of data processing products connected by communications lines for exchanging information between stations.

network address. An address that identifies a specific host on a network. Synonymous with *host ID*.

network data. RPC: Data represented in a format defined by a transfer syntax. See also *transfer syntax*.

Network Data Representation (NDR). RPC: The transfer syntax defined by the Network Computing Architecture. See *transfer syntax*.

network protocol. A communications protocol from the Network Layer of the Open Systems Interconnection (OSI) network architecture, such as the Internet Protocol (IP).

Network Time Protocol (NTP). A clock synchronization protocol commonly used on an Internet.

node. (1) An endpoint of a link, or a junction common to two or more links in a network. Nodes can be preprocessors, controllers, or workstations, and they can vary in routing and other functional capabilities. (2) In network topology, the point at an end of a branch. It is usually a physical machine.

null time provider. The daemon that fetches the time from the hardware clock of the DCE host for DTS.

NSI. Name service interface.

NSI binding attribute. RPC: An RPC-defined attribute (NSI attribute) of a name service entry; the binding attribute stores binding information for one or more interface identifiers offered by an RPC server and identifies the entry as an RPC server entry. See *binding information* and *NSI object attribute*. See also *server entry*.

NSI group attribute. RPC: An RPC-defined attribute (NSI attribute) of a name service entry that stores the entry names of the members of an RPC group and identifies the entry as an RPC group. See *group*.

NSI object attribute. RPC: An RPC-defined attribute (NSI attribute) of a name service entry that stores the object UUIDs of a set of RPC objects. See *object*.

NSI profile attribute. RPC: An RPC-defined attribute (NSI attribute) of a name service entry that stores a collection of RPC profile elements and identifies the entry as an RPC profile. See *profile*.

NTP. Network Time Protocol.

NULL. In the C language, a pointer that does not point to a data object.

O

object. (1) A data structure that implements some feature and has an associated set of operations. (2) RPC: For RPC applications, anything that an RPC server defines and identifies to its clients using an object Universal Unique Identifier (UUID). An RPC object is often a physical computing resource such as a database, directory, device, or processor. Alternatively, an RPC object can be an abstraction that is meaningful to an application, such as a service or the location of a server. See *object UUID*. (3) XDS: Anything in the world of telecommunications and information processing that can be named and for which the directory information base (DIB) contains information. (4) XOM: Any of the complex information objects created, examined, changed, or destroyed by means of the interface.

object class. GDS, CDS: An identified family of objects that share certain characteristics. An object class can be specific to one application or shared among a group of applications. An application interprets and uses an entry's class-specific attributes based on the class of the object that the entry describes.

object class table (OCT). A recurring attribute of the directory schema with the description of the object classes permitted.

object entry. CDS: The name of a resource (such as a node, disk, or application) and its associated attributes, as stored by CDS. CDS administrators, client application users, or the client applications themselves can give a resource an object name. CDS supplies some attribute information (such as a creation timestamp) to become part of the object, and the client application may supply more information for CDS to store as other attributes. See *entry*.

object identifier (OID). A value (distinguishable from all other such values) that is associated with an information object. It is formally defined in the CCITT X.208 standard.

object management (OM). The creation, examination, change, and deletion of potentially complex information objects.

object name. CDS: A name for a network resource.

object UUID. RPC: The Universal Unique Identifier (UUID) that identifies a particular RPC object. A server specifies a distinct object UUID for each of its RPC objects. To access a particular RPC object, a client uses the object UUID to find the server that offers the object. See *object*.

OCT. Object class table.

OID. Object identifier.

OM. Object management.

opaque. A datum or data type whose contents are not visible to the application routines that use it.

Open Software Foundation (OSF). A nonprofit research and development organization set up to encourage the development of solutions that allow computers from different vendors to work together in a true open-system computing environment.

open systems interconnection (OSI). The interconnection of open systems in accordance with standards of the International Organization for Standardization (ISO) for the exchange of information.

operation. (1) GDS: Processing performed within the directory to provide a service, such as a read operation.

(2) RPC: The task performed by a routine or procedure that is requested by a remote procedure call.

organization. (1) The third field of a subject identifier. (2) Security: Data that associates a named set of users who can be granted common access rights that are usually associated with administrative policy.

OSF. Open Software Foundation.

OSI. Open systems interconnection

P

PAC. Privilege attribute certificate.

package. XOM: A specified group of related object management (OM) classes, denoted by an object identifier.

packet. (1) In data communication, a sequence of binary digits, including data and control signals, that is transmitted and switched as a composite whole. [1] The data, call control signals, and error control information are arranged in a specific format. (2) See *call-accepted packet, call-connected packet, call-request packet*. See *clear-confirmation packet, clear-indication packet, clear-request packet*. See *data packet, incoming-call packet.*

parent directory. CDS: Any directory that has one or more levels of directories beneath it in a cell namespace. A directory is the parent of any directory immediately beneath it in the hierarchy.

parent process. A process created to carry out a program. The parent process in turn creates child processes to process requests. Contrast with *child process*.

partially bound binding handle. RPC: A server binding handle that contains an incomplete server address lacking an endpoint. Contrast with *fully bound binding handle*.

Partitioned data set (PDS). A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

password. A secret string of characters shared between a computer system and a user. The user must specify the character string to gain access to the system.

PCS. Portable Character Set.

PDS. Partitioned data set

peer trust. A type of trust relationship established between two cells by means of a secret key shared by authentication surrogates maintained by the two cells. A peer trust relationship enables principals in one cell to communicate securely with principals in the other.

permission. (1) The modes of access to a protected object. The number and meaning of permissions with respect to an object are defined by the access control list (ACL) Manager of the object. (2) GDS: One of five groups that assigns modes of access to users: MODIFY PUBLIC, READ STANDARD, MODIFY STANDARD, READ SENSITIVE, or MODIFY SENSITIVE. Synonymous with *access right*. See also *access control list*.

person. See *principal*.

pickle. A type of data encoding. When a Remote Procedure Call (RPC) sends data between a client and a server, it serializes the user's data structures by using the IDL Encoding Services (ES). This serialization scheme for encoding and decoding data is informally called *pickling*.

ping. TCP/IP: Utility in TCP/IP which is used to test if a destination host can be reached by sending test packets and waiting for a reply. RPC: In the RPC control program, a command to test if a server is listening.

pipe. (1) RPC: A mechanism for passing large amounts of data in a remote procedure call. (2) The data structure that represents this mechanism.

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

platform. The operating system environment in which a program runs.

PLT. Program list table.

port. (1) Part of an Internet Protocol (IP) address specifying an endpoint. (2) To make the programming changes necessary to allow a program that runs on one type of computer to run on another type of computer.

Portable Character Set. A set of characters to enable internationalization. A character set used by DCE to enable word wide connectivity by ensuring that a minimum group of characters is supported in DCE. All DCE RPC clients and servers are required to support the DCE PCS.

position (within a string). XOM: The ordinal position of one element of a string relative to another.

position (within an attribute). XOM: The ordinal position of one value relative to another.

presentation address. An unambiguous name that is used to identify a set of presentation service access points. Loosely, it is the network address of an open systems interconnection (OSI) service.

primary name. The string name of an object to which any aliases for that object refer. The DCE refers to objects by their primary names, although DCE users may refer to them by their aliases.

principal. Security: An entity that can communicate securely with another entity. In the DCE, principals are represented as entries in the Registry database and include users, servers, computers, and authentication surrogates.

privacy. RPC: A protection level that encrypts RPC argument values. in secure RPC communications.

privilege attribute. Security: An attribute of a principal that may be associated with a set of permissions. DCE privilege attributes are identity-based and include the principal's name, group memberships, and local cell.

privilege attribute certificate (PAC). Security: Data describing a principal's privilege attributes that has been certified by an authority. In the DCE, the Privilege Service is the certifying authority; it seals the privilege attribute data in a ticket. The authorization protocol, DCE Authorization, determines the permissions granted to principals by comparing the privilege attributes in PACs with entries in an access control list.

privilege service. Security: One of three services provided by the Security Service; the Privilege Service certifies a principal's privileges. The other services are the Registry Service and the Authentication Service.

privilege ticket. Security: A ticket that contains the same information as a simple ticket, and also includes a privilege attribute certificate. See *service ticket*, *simple ticket*, and *ticket-granting ticket*.

profile. RPC: An entry in a name service database that contains a collection of elements from which name service interface (NSI) search operations construct search paths for the database. Each search path is composed of one or more elements that refer to name service entries corresponding to a given RPC interface and, optionally, to an object. See *NSI profile attribute* and *profile element*.

profile element. RPC: A record in an RPC profile that maps an RPC interface identifier to a profile member (a server entry, group, or profile in a name service database). See *profile*. See also *group*, *interface identifier* and *server entry*.

profile member. RPC: A name service entry whose name occupies the member field of an element of the profile. See *profile*.

program list table (PLT). CICS/ESA®: A data area that contains a list of programs to be invoked.

programming interface. The supported method through which customer programs request software services. The programming interface consists of a set of callable services provided with the product.

protection level. The degree to which secure network communications are protected. Synonymous with *authentication level*.

protocol. A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication.

protocol family. Synonym for *address family*.

protocol sequence. Synonym for *RPC protocol sequence*.

R

RACF. Resource Access Control Facility.

RDN. Relative distinguished name.

read access. CDS: An access right that grants the ability to view data.

read-only replica. (1) CDS: A copy of a CDS directory in which applications cannot make changes. Although applications can look up information (read) from it, they cannot create, change, or delete entries in a read-only replica. Read-only replicas become consistent with other, changeable replicas of the same directory during skulks and routine propagation of updates. (2) Security: A replicated Registry server.

realm. Security: A cell, considered exclusively from the point of view of Security; this term is used in Kerberos specifications. The term cell designates the basic unit of DCE configuration and administration and incorporates the notion of a realm.

referral. GDS: An outcome that can be returned by a DSA that cannot perform an operation itself. The referral identifies one or more other DSAs more able to perform the operation.

register. (1) RPC: To list an RPC interface with the RPC runtime. (2) To place server-addressing information into the local endpoint map. (3) To insert authorization and authentication information into binding information. See *endpoint map* and *RPC interface*.

Registry database. Security: A database of security information about principals, groups, organizations, accounts, and security policies.

Registry replica. Security: A read-only instance of a Registry database.

Registry Service. Security: One of three services provided by the Security Service; the Registry Service manages information about principals, accounts, and security policies. The other services are the Privilege Service and the Authentication Service.

relative distinguished name (RDN). GDS, XDS: A set of Attribute Value Assertions (AVAs).

relative time. A discrete time interval that is usually added to or subtracted from an absolute time. See *absolute time*.

remote. Pertaining to a device, file or system that is accessed by your system through a communications line. Contrast with *local*.

remote procedure. RPC: An application procedure located in a separate address space from calling code. See *remote procedure call*.

remote procedure call. RPC: A client request to a service provider located anywhere in the network.

Remote Procedure Call (RPC). A DCE component. It allows requests from a client program to access a procedure located anywhere in the network.

replica. CDS: A directory in the CDS namespace. The first instance of a directory in the namespace is the master replica. See *master replica* and *read-only replica*.

replica set. CDS: The set of all copies of a CDS directory. Information about a directory's replica set is contained in an attribute of directories and child pointers called **CDS_Replicas**. The attribute contains the type of each replica (master or read-only) and the clearinghouse where it is located. When skulking a directory, CDS refers to the directory's replica set to ensure that it finds all copies of that directory. During a lookup, CDS may refer to the replica set in a child pointer when trying to locate a directory that does not exist in the local clearinghouse.

replication. The making of a shadow of a database to be used by another node. Replication can improve availability and load-sharing.

request. A command sent to a server over a connection.

resource. Items such as printers, plotters, data storage, or computer services. Each has a unique identifier associated with it for naming purposes.

Resource Access Control Facility (RACF). An IBM licensed program, that provides for access control by identifying and verifying the users to the system, authorizing access to protected resources, and logging the detected unauthorized access to protected resources.

return value. A function result that is returned in addition to the values of any output or input/output arguments.

ROM. Read-only memory.

RPC. Remote Procedure Call.

RPC control program (RPCCP). An interactive administrative facility for managing name service entries and endpoint maps for RPC applications.

RPCCP. RPC control program

RPC interface. A logical group of operations, data types, and constant declarations that serves as a network contract for a client to request a procedure in a server. See also *interface definition* and *operation*.

RPC protocol. An RPC-specific communications protocol that supports the semantics of the DCE RPC API and runs over either connectionless or connection-oriented communications protocols.

RPC protocol sequence. A valid combination of communications protocols represented by a character string. Each RPC protocol sequence typically includes three protocols: a network protocol, a transport protocol, and an RPC protocol that works with the network and transport protocols. See *network protocol*, *RPC protocol*, and *transfer protocol*. Synonymous with *protocol sequence*.

RPC runtime. A set of operations that manages communications, provides access to the name service database, and performs other tasks, such as managing servers and accessing security information, for RPC applications. See *RPC runtime library*.

RPC runtime library. A group of routines of the RPC runtime that support the RPC applications on a system. The runtime library provides a public interface to application programmers, the application programming interface (API), and a private interface to stubs, the stub programming interface (SPI). See *RPC runtime*.

RPC thread. A logical thread within which a remote procedure call is run. See *thread*.

S

schema. See *directory schema*.

secret key. Security: A long-lived encryption key shared between a principal and the Authentication Service.

Security Service. A DCE component that provides trustworthy identification of users, secure communications, and controlled access to resources in a distributed system.

segment. One or more contiguous elements of a string.

server. (1) On a network, the computer that contains programs, data, or provides the facilities that other computers on the network can access. (2) The party that receives remote procedure calls. Contrast with *client*.

server application thread. RPC: A thread running the server application code that initializes the server and listens for incoming calls. See *application thread*, *client application thread*, *local application thread*, and *RPC thread*.

server binding information. RPC: Binding information for a particular RPC server. See *binding information* and *client binding information*.

server entry. RPC: A name service entry that stores the binding information associated with the RPC interfaces of a particular RPC server and object Universal Unique Identifiers (UUIDs) for any objects offered by the server. See also *binding information*, *NSI binding attribute*, *NSI object attribute*, *object* and *RPC interface*.

server stub. RPC: The surrogate calling code for an RPC interface that is linked with server application code containing one or more sets of remote procedures (managers) that implement the interface. See *client stub*. See also *manager*.

service. In network architecture, the capabilities that the layers closer to the physical media provide to the layers closer to the end user.

service ticket. Security: A ticket for a specified service other than the ticket-granting service. See *privilege ticket*, *simple ticket*, and *ticket-granting ticket*.

session. GDS: A sequence of directory operations requested by a particular user of a particular directory user agent (DUA) using the same session object management (OM) object.

session key. Security: A short-lived encryption key provided by the Authentication Service to two principals for the purpose of ensuring secure communications between them. Synonymous with *conversation key*.

shell script. A file containing shell commands. If the file can be processed, you can specify its name as a simple command. Processing of a shell script causes a shell to run the commands in the script. Alternatively, a shell can be requested to run the commands in a shell script by specifying the name of the shell script as the operand **sh** utility.

signal. Threads: To wake only one thread waiting on a condition variable. See *broadcast*.

signed. Security: Pertaining to information that is appended to an enciphered summary of the information. This information is used to ensure the integrity of the data, the authenticity of the originator, and the unambiguous relationship between the originator and the data.

sign-on. (1) A procedure to be followed at a terminal or workstation to establish a link to a computer. (2) To begin a session at a workstation. (3) Same as log on or log in.

simple name. CDS: One element in a CDS full name. Simple names are separated by slashes in the full name.

simple ticket. Security: A ticket that contains the principal's identity, a session key, a timestamp and other information, sealed using the target's secret key. See *privilege ticket*, *service ticket*, and *ticket-granting ticket*.

single sign-on. In OS/390 DCE, single sign-on to DCE allows an OS/390 user who has already been authenticated to an external security manager, such as RACF, to be logged in to DCE. DCE does this automatically when a DCE application is started, if the user is not already logged in to DCE.

skew. The time difference between two clocks or clock values.

skulk. CDS: A process by which CDS makes the data consistent in all replicas of a particular directory. CDS collects all changes made to the master replica since the last skulk was completed, and disseminates the changes from the up-to-date replica to all other existing replicas of the directory. All replicas of a directory must be available for a skulk to be considered successful.

socket. A unique host identifier created by the concatenation of a port identifier with a TCP/IP address.

soft link. CDS: A pointer that provides an alternative name for an object entry, directory, or other soft link in the namespace. A soft link can be permanent or it can expire after a specific period of time. The CDS server also can delete it after the name that the link points to is deleted.

specific. XOM: The attribute types that can appear in an instance of a given class, but not in an instance of its superclasses.

standard. A model that is established and widely used.

string. An ordered sequence of bits, octets, or characters, accompanied by the string's length.

structure rule table (SRT). GDS: A recurring attribute of the directory schema with the description of the permitted structures of distinguished names.

stub. RPC: A code module specific to an RPC interface that is generated by the Interface Definition Language (IDL) compiler to support remote procedure calls for the interface. RPC stubs are linked with client and server applications and hide the intricacies of remote procedure calls from the application code. See *client stub* and *server stub*.

subject identifier (SID). A string that identifies a user or set of users. Each SID consists of three fields in the form person.group.organization. In an account, each field must have a specific value; in an access control list (ACL) entry, one or more fields may use a wildcard.

subordinate. GDS, XDS: In the directory information tree (DIT), an entry whose distinguished name includes that of the other as a prefix.

synchronization. DTS: The process by which a Distributed Time Service entity requests clock values from other systems, computes a new time from the values, and adjusts its system clock to the new time.

synchronization list. DTS: The list of servers that a DTS entity has discovered. The entity sends requests for clock values to the servers on the list.

syntax. (1) XOM: An object management (OM) syntax is any of the various categories into which the OM specification statically groups values on the basis of their form. These categories are additional to the OM type of the value. (2) A category into which an attribute value is placed on the basis of its form. See *attribute syntax*.

sysplex. Systems complex. Multiple OS/390 systems connected together to perform the processing for an installation.

system time. The time value maintained and used by the operating system.

T

TCP. Transmission Control Protocol

TCP/IP. Transmission Control Protocol/Internet Protocol

TDF. Time differential factor.

thread. A single sequential flow of control within a process.

thread handle. RPC: A data item that enables threads to share a storage management environment.

Threads Service. A DCE component that provides portable facilities that support concurrent programming. The threads service includes operations to create and control multiple threads of execution in a single process and to synchronize access to global data within an application.

ticket. Security: An application-transparent mechanism that transmits the identity of an initiating principal to its target. See *privilege ticket*, *service ticket*, *simple ticket* and *ticket-granting ticket*.

ticket-granting ticket. Security: A ticket to the ticket-granting service. See *privilege ticket*, *service ticket*, and *simple ticket*.

time differential factor (TDF). DTS: The difference between universal time coordinated (UTC) and the time in a particular time zone.

time provider (TP). DTS: A process that queries universal time coordinated (UTC) from a hardware device and provides it to the server.

time provider interface (TPI). An interface between the DTS server and external time provider process. The DTS server uses the interface to communicate with the time provider and to obtain timestamps from an external time source.

time provider program. DTS: An application that functions as a time provider.

tower. CDS: A set of physical address and protocol information for a particular server. CDS uses this information to locate the system on which a server resides and to determine which protocols are available at the server. Tower values are contained in the **CDS_Towers** attribute associated with the object entry that represents the server in the cell namespace.

TP. Time provider.

TP server. DTS: A server connected to a time provider (TP).

TPI. Time provider interface.

transaction. (1) A unit of processing consisting of one more application programs initiated by a single request, often from a terminal. (2) IMS/ESA®: A message destined for an application program.

transfer syntax. RPC: A set of encoding rules used for transmitting data over a network and for converting application data to and from different local data representations. See also *Network Data Representation*.

Transmission Control Protocol (TCP). A communications protocol used in Internet and any other network following the U.S. Department of Defense standards for inter-network protocol. TCP provides a reliable host-to-host protocol in packet-switched communication networks and in an interconnected system of such networks. It assumes that the Internet Protocol is the underlying protocol. The protocol that provides a reliable, full-duplex, connection- oriented service for applications.

Transmission Control Protocol/Internet Protocol (TCP/IP). A set of non-proprietary communications protocols that support peer-to-peer connectivity functions for both local and wide area networks.

transport layer. A network service that provides end-to-end communications between two parties, while hiding the details of the communications network. The Transmission Control Protocol (TCP) and International Organization for Standardization (ISO) TP4 transport protocols provide full-duplex virtual circuits on which delivery is reliable, error free, sequenced, and duplicate free. User Datagram Protocol (UDP) provides no guarantees. The connectionless RPC protocol provides some guarantees on top of UDP.

Trivial File Transfer Protocol (TFTP). Transfers files between hosts using minimal protocol.

trust peer. One side of a cross-registration that enables two cells to have peer trust. See *peer trust*.

type. XOM: A category into which attribute values are placed on the basis of their purpose. See *attribute type*.

type UUID. RPC: The Universal Unique Identifier (UUID) that identifies a particular type of object and an associated manager. See also *manager* and *object*.

U

UDP. User Datagram Protocol.

unexport. RPC: To remove binding information from a server entry in a name service database. Contrast with *export*.

Universal Time Coordinated (UTC). The basis of standard time throughout the world. Synonymous with Greenwich mean time (GMT).

Universal Unique Identifier (UUID). RPC: An identifier that is immutable and unique across time and space. A UUID can uniquely identify an entity such as an object or an RPC interface. See *interface UUID*, *object UUID*, and *type UUID*.

update propagation. CDS: An immediate attempt to apply a change to all replicas of the CDS directory in which the change was just made. An update propagation delivers changes in a more efficient and timely way than a skulk.

update timestamp (UTS). CDS: An attribute that identifies the time at which the most recent change was made to any attribute of a particular CDS name. For directories, the UTS reflects changes made only to attributes that apply to the actual directory (not one of its replicas).

user. A person who requires the services of a computing system.

User Datagram Protocol (UDP). In TCP/IP, a packet-level protocol built directly on the Internet protocol layer. UDP is used for application-to-application programs between TCP/IP host systems.

UTC. Universal Time Coordinated

UTS. Update timestamp.

UUID. Universal unique identifier

V

value. XOM: An arbitrary and complex information item that can be viewed as a characteristic or property of an object. See *attribute value*.

vendor. Supplier of software products.

Virtual Telecommunications Access Method (VTAM®). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

VTAM. Virtual Telecommunications Access Method.

W

WAN. Wide area network.

well-known endpoint. RPC: A preassigned, stable endpoint that a server can use every time it runs. Well-known endpoints typically are assigned by a central authority responsible for a transport protocol. An application declares a well-known endpoint either as an attribute in an RPC interface header or as a variable in the server application code. See *dynamic endpoint* and *endpoint*.

wide area network (WAN). A network that provides communication services to a geographic area larger than that served by a local area network (LAN).

WLB. Workload Balancing

WLM. Workload Management

Workload Balancing (WLB). The task used in a Parallel Sysplex subsystem with DCE Release 6 or higher on OS/390. Workload Balancing allows DCE servers that export the same interfaces within an OS/390 Sysplex to have their work distributed evenly to them from their corresponding DCE clients.

Workload Management (WLM). The service available on OS/390, used in a Parallel Sysplex subsystem. Workload Management allows a system programmer to provide a Coupled Data Set (CDS) along with a set of service policies describing the amount of resource that the Sysplex is able to supply to do work. For DCE Workload Balancing, the WLM product must be configured and the WLM subsystem must be operating in "goal mode".

X

X.500. The CCITT/ISO standard for the open systems interconnection (OSI) application-layer directory. It allows users to register, store, search, and retrieve information about any objects or resources in a network or distributed system.

XDS. The X/OPEN Directory Service API.

X/OPEN Directory Service (XDS). An application program interface that DCE uses to access its directory service components. XDS provides facilities for adding, deleting, and looking up names and their attributes. The XDS library detects the format of the name to be looked up and directs the calls it receives to either GDS or CDS. XDS uses the X/OPEN object management (XOM) API to define and manage its information.

X/OPEN object management (XOM). An interface for creating, deleting, and accessing objects containing information. It is an object-oriented architecture: Each object belongs to a particular class, and classes can be derived from other classes inheriting the characteristics of the original classes, The representation of the object is transparent to the programmer; the object can be manipulated only through the XOM interface.

XOM. The X/OPEN Object Management API.

Bibliography

This bibliography is a list of publications for OS/390 DCE and other products. The complete title, order number, and a brief description is given for each publication.

OS/390 DCE Publications

This section lists and provides a brief description of each publication in the OS/390 DCE library.

Overview

- *Distributed Computing Environment: Understanding the Concepts*, GC09-1478

  This book introduces Open Software Foundation (OSF) DCE. It describes the technology components of DCE, from a high-level overview to a discussion of the interdependencies among the components.

- *OS/390 DCE Introduction*, GC28-1581

  This book introduces OS/390 DCE. Whether you are a system manager, technical planner, OS/390 system programmer, or application programmer, it will help you understand DCE, and evaluate the uses and benefits of including OS/390 DCE as part of your information processing environment.

Planning

- *OS/390 DCE Planning*, SC28-1582

  This book helps you plan for the organization and installation of OS/390 DCE. It discusses the benefits of distributed computing in general, and describes how to develop plans for a distributed system in an OS/390 DCE environment.

Administration

- *OS/390 DCE Configuring and Getting Started*, SC28-1583

  This book helps system and network administrators configure OS/390 DCE.

- *OS/390 DCE Administration Guide*, SC28-1584

  This book helps system and network administrators understand OS/390 DCE, and tells how to administer it from the batch, TSO, and shell environments.

- *OS/390 DCE Command Reference*, SC28-1585

  This book provides reference information for the commands that system and network administrators use to work with OS/390 DCE.

- *OS/390 OpenEdition DCE User's Guide*, SC28-1586

  This book describes how to use OS/390 DCE to work with your user account, use the directory service, work with namespaces, and change access to objects that you own.

Application Development

- *OS/390 DCE Application Development Guide: Introduction and Style*, SC28-1587

  This book assists you in designing, writing, compiling, linking, and running distributed applications in OS/390 DCE.

- *OS/390 DCE Application Development Guide: Core Components*, SC28-1588

  This book assists programmers in developing applications using application facilities, threads, remote procedure calls, distributed time service, and security service.

- *OS/390 DCE Application Development Guide: Directory Services*, SC28-1589

  This book describes the OS/390 DCE directory service and assists programmers in developing applications for the cell directory service and the global directory service.

- *OS/390 DCE Application Development Reference*, SC28-1590

  This book explains the DCE Application Program Interfaces (APIs) that you can use to write distributed applications on OS/390 DCE.

Reference

- *OS/390 DCE Messages and Codes*, SC28-1591

  This book provides detailed explanations and recovery actions for the messages, status codes, and exception codes issued by OS/390 DCE.

OS/390 Security Server Publications

This section lists and provides a brief description of books in the OS/390 Security Server library that may be needed for the OS/390 DCE Security Server and for RACF® interoperability.

- *OS/390 DCE Security Server Overview*, GC28-1938

  This book describes the DCE security server and provides a road map for DCE security server information in the OS/390 DCE library.

- *OS/390 Security Server (RACF) Security Administrator's Guide*, SC28-1915.

  This book explains RACF concepts and describes how to plan for and implement RACF.

- *OS/390 Security Server LDAP Server Administration and Usage Guide*, SC24-5861

  This book describes how to install, configure, and run the stand-alone LDAP daemon (SLAPD). It is intended for administrators who will maintain the server and database.

- *OS/390 Security Server LDAP Client Application Development Guide and Reference*, SC24-5878

  This book describes the Lightweight Directory Access Protocol (LDAP) client APIs that you can use to write distributed applications on OS/390 DCE and gives you information on how to develop LDAP applications.

- *Firewall Technologies Guide and Reference*, SC24-5835

  This book provides the configuration, commands, messages, examples and problem determination for the OS/390 Firewall Technologies. It is intended for network or system security administrators who install, administer and use the OS/390 Firewall Technologies.

Tool Control Language Publication

- *Tcl and the Tk Toolkit*, John K. Osterhout, (c)1994, Addison—Wesley Publishing Company.

  This non-IBM book on the Tool Control Language is useful for application developers, DCECP script writers, and end users.

IBM C/C++ Language Publication

- *IBM OS/390 C/C++ Programming Guide*, SC09-2362

  This book describes how to develop applications in the C/C++ language in OS/390.

OS/390 DCE Application Support Publications

This section lists and provides a brief description of each publication in the OS/390 DCE Application Support library.

- *OS/390 DCE Application Support Configuration and Administration Guide*, SC24-5834

  This book helps system and network administrators understand and administer Application Support.

- *OS/390 DCE Application Support Programming Guide*, SC24-5833

  This book provides information on using Application Support to develop applications that can access CICS® and IMS™ transactions.

Encina Publications

- *OS/390 Encina Toolkit Executive Guide and Reference*, SC24-5832

  This book discusses writing Encina applications for OS/390.

- *OS/390 Encina Transactional RPC Support for IMS*, SC24-5874

  This book is to help software designers and programmers extend their IMS transaction applications to participate in a distributed, transactional client/server application.

Index

Special Characters

/.:/fs junction 171
/.:/sec junction 171

A

Abbreviations 49
ABEND, automatic restart 210
access control
  in CDS 180
  namespace 22
  setting up in new namespace 204
access control list (ACL)
  authorization and 307
  definition 6, 197
access control lists (ACLs), in DTS 287
access control lists (ACLs), inheritance 220
access control lists (ACLs), permissions for krbtgt directory 375
access control lists (ACLs), registry objects 441
access, denying 319
account (user) expiration 356
account information summary
account lifespan policy 411
accounts
  creating and maintaining 347
  cross-cell authentication 377, 379
  for principals 347
  in registry database 298
  machine 349
  permissions required
    to add 433
    to change 439
    to delete 436
  server
    creating 348
    passwords for 348
    why servers need accounts 347
  user 347
accounts, about 347
accounts, changing 355
accounts, changing passwords 439
accounts, changing registry information 439
accounts, creating 351, 433, 436
accounts, deleting 355, 436
accounts, displaying registry information 381, 382
accounts, expiration information 350
accounts, for foreign cells 375, 379
accounts, lifespan 411
accounts, machine 349, 357, 361
accounts, managing 23
accounts, membership lists 344
accounts, permissions 436
accounts, reserved 332, 333
accounts, server 348, 357, 361
accounts, summary of information 352
accounts, user 347
accounts. server 347
ACL (access control list)
  defined 197
  description 307
  entry types 311
  extended entry type 315
  initial ACLs for containers created in containers 323
  initial ACLs for objects created in containers 321
  initial registry 443
  overview 307
  types 197
ACL Editor
  administration tool 14
  description 23
ACL entry types 311, 316
  CDS, supported by 198
  principals and groups 310
  RPC ACL Manager, supported by 149
  unauthenticated 205
  valid for ACL managers 442
ACL entry types, compared to ACL types 321
ACL entry types, in future DCE releases 315
ACL entry, defined 197
ACL facility 297
ACL manager 442
ACL Manager, for registry database 442
ACL Manager, role in checking sequence 316, 319
ACL Manager, role in granting access 308
ACL Manager, scope of support 308
ACL Manager, support for entry types 315
ACL, dts_audit_events 555
ACLs, checking sequence 316, 319
ACLs, components and scope of entries 310
ACLs, control programs for managing 319
ACLs, copying to other objects 320
ACLs, default 321, 325
ACLs, denying access 319
ACLs, displaying 386
ACLs, function 307
ACLs, keys 310
ACLs, scope 307
ACLs, scope compared to UNIX permission bits 308
ACLs, types of, editing 321

administer permission 180, 200
administration
 commands 29
 DCE, OS/390 considerations 29
administration scenarios for RACF utilities 400
administration tools
 ACL Editor 14, 23
 availability 18
 CDS control program 9
 DTS control program 13
 Registry Editor 14
administrators 375
advertise command 270
alias 332, 336, 341, 345
aliases, changing 345
aliases, creating 345
any_other entry type 312
any_other_entry type 313
applications, client 173
architecture, DCE 3
arguments, entering in ISPF 30
ASCII, conversion to EBCDIC 181
attribute name, mapping to object identifiers 535
Attribute schema, defined 363
attribute type, GDS 168
Attribute types, access control 363
attribute value assertion (AVA) 168
attribute value, GDS 168
attributes
 CDS_Convergence 191
 CDS_GDAPointers 251
 CDS_Replicas 186, 188
 class-specific 176
 description 173
 object 9
aud_audit_events 553
audit 445
Audit clients, definition 446
Audit code point 446
Audit daemon 457
Audit daemon, attributes 463
Audit daemon, controlling access to 458
Audit daemon, giving permissions to 458
Audit daemon, initial ACL 458
Audit daemon, starting 457
Audit event 446
Audit event class 446
Audit event class file 447
Audit event class file, SEP line 459
Audit event class name 447
Audit event class number 447
Audit event class number, format 448
Audit event class, how to define 459
Audit event number 446
Audit filter 448
Audit filter rules, high-water-mark rule 450
Audit filter rules, override rule 450
Audit filter, audit actions 449
Audit filter, audit condition 449
Audit filter, creating and maintaining 460
Audit filter, guides 449
Audit filter, key 449
Audit filter, subject identity 449
Audit filter, types 449
Audit Service 445
Audit Service, ACL permissions 458
Audit Service, administration tasks 453, 457
Audit Service, application programmer's tasks 452
Audit Service, components 445
Audit Service, concepts 445
Audit Service, features 445
audit services, auditable events 553
Audit trail file 451
Audit trail file, changing storage option 464
Audit trail file, controlling 464
Audit trail file, displaying 463
auditable events, audit services 553
auditable events, security services 559
auditable events, time services 555
authentication information 432
authentication policies 350, 413, 432
Authentication Service 14, 297
Authentication Service, how it works 349, 351
Authentication Service, shared authentication keys 375
authentication services, rpc_c_authn_dce_secret 560
authentication, effect on CDS 179
authentication, managing 337
authentication, preauthentication 336
authorization
 CDS overview 197
 effect on CDS 180
authorization group, creating 205
automatic restart, CDS 210
automatic single sign-on to DCE 410

B

background skulk time 192
backing up files, utilities for 42
backup couriers 284
backup procedure
 directory replication 217
 for namespace information 217
 traditional operating system backup, CDS 217
backup strategy 42
batch, running commands in 31
Berkeley Internet Naming Domain (BIND) 170
bibliography 591

BIND (Berkeley Internet Naming Domain) 170
BNF, rules 539
books, list of DCE and related 591

C

cache entries, removing stale 210
cache, changing size 211
caching
- definition 6
- results of lookups 173

case-matching rules, GDS 537
CDS
- backup procedures 217
- benefits 173
- components 173, 174
- contacting another cell 12
- control program 20
- interaction with GDA and GDS 165
- lookup 174
- maintenance tasks 20
- managing namespaces 21
- managing servers 21
- monitoring 20
- overview 9
- permissions, definition 200
- requesting names in another cell 11
- sample configuration 174
- security 180
- user interaction 162
- user interaction with 163

CDS advertiser
- functions of 177
- in OS/390 DCE 179
- in OSF DCE 178
- restarting 209
- starting and stopping 38

CDS cache file, protecting the 181
CDS cache, recovering from a corrupted 211
CDS cached clearinghouse counters, monitoring 207
CDS clerk
- CDS in OSF DCE 178
- counters 207
- finding a GDA 252
- finding a name 189
- in OS/390 DCE 179
- on a LAN 174
- restarting 209
- starting and stopping 38

CDS clerks, clearinghouse communications 207
CDS clerks, disabling 208
CDS clerks, interaction with GDA 254
CDS clerks, viewing counters 207
CDS control program
- permissions required to run 201

CDS daemon
- starting and stopping 209

CDS directory, top-level 500
CDS globalnames file 536
CDS hosts directory 503
CDS names, controlling access 197
CDS server
- components of 177
- how clerks learn about 187, 188

CDS servers, disabling 208
CDS servers, permissions for 203
CDS servers, viewing counters 207
CDS subsys directory 506
CDS_attributes file 535
CDS_attributes file, modifying 538
CDS_Convergence attribute 191
CDS_GDAPointers attribute 251
CDS_globalnames 536
CDS_globalnames file
- case-matching rules 537
- description 535
- modifying 538

CDS_LinkTarget attribute 240
CDS_Replicas attribute 186, 188
CDS, finding names 187
CDS, looking up names 183
CDS, propagating changes 191
CDS, updating data 191
CDS, user interfaces 181
cdscache show command 207
cdscp commands 195
cdscp commands, required permissions 203
cdscp commands, show 229
cdscp, administration tool 9
cdscp, commands 195
cdscp, description 194
cdsd file system full condition, recovering from 212
cell
- description 5, 299
- effects on naming 165
- root 167

Cell Directory Service (CDS) concepts 182
Cell Directory Service (CDS), appending directories 233
Cell Directory Service (CDS), deleting non-replicated directories 245
Cell Directory Service (CDS), deleting replicas 245
Cell Directory Service (CDS), displaying attribute values 227, 229
Cell Directory Service (CDS), listing contents of directories 227
Cell Directory Service (CDS), merging directories 232
Cell Directory Service (CDS), restoring merged directories 236

Cell Directory Service (CDS), user interfaces 181
Cell Directory Service (CDS), viewing the namespace 227, 229
cell directory service namespace 499
cell entry
  creating in DNS 255
cell name
  DNS style 165, 171
  obtaining 254
cell namespace
  definition 167
  description 5
  initial DCE configuration 499
  monitoring 20
  security 22
cell profile 270
cell root directory 167
Cell Service Profile, global-set membership 555
cell-relative names 166
cells, access between 375, 379
changes, how CDS propagates 191, 192
changing size of cache 211
child directory 167
child pointer
  description 188
  relationship to directory 186
  role in lookups 189
child pointers 177, 187
class-specific attribute 176
class, object 176
clearinghouse
  counters 207
  definition 174
  diagram 177
  displaying contents 208
  how CDS finds 176
  management tasks 207
  monitoring counters 207
  preferred 229
  preserving across system upgrade 211
clearinghouse object entry 176
clearinghouse object entry, diagram 184
clearinghouse show command 208
clearinghouses, communications with CDS clerks 207
clearinghouses, deleting 249, 250
clearinghouses, relocating 246, 249
clearinghouses, viewing counters 207
clerk
  cache 173
  controlling access 200
  description 173
  disabling 209
  DTS, description 259, 268
  management tasks 207
  monitoring contact with clearinghouses 207
clerk *(continued)*
  monitoring counters 207
client 3, 259
client applications 173
client/server interaction 4
client/server model 3, 173
clock adjustment 265
clock error 262
clock set command 286
clocks, correcting malfunctions 281, 282
clocks, forcing synchronization 287
clocks, restricting synchronization cycles 280
code point 446
command input and output redirection 32
command interface 182
commands
  DCE administration 29
  for determining if server is listening 120
  user 29
commands, dcecp 203, 271
Commands, login_activity 356
communications 262
communications delay 262
components
  interaction between 15
  overview 3, 7
computed time 259, 263
computed time, finding 263
configuration file, daemon 37
configuration, password management server 340
container ACL, default example 323
containers, ACL editing 321
containers, default ACLs 322
containers, definition of 321
containers, initial access control lists 321
control permission 180, 200
control program 182
convergence, CDS directory 225
conversion
  from ASCII to EBCDIC 181
  from EBCDIC to ASCII 181
conversion, DTS time 539
Coordinated Universal Time (UTC) 259
correctness 259
counters, CDS clerk, server, and clearinghouse 207
courier, backup 270
couriers, backup 284
couriers, designating 284, 285
couriers, DTS
  description 270
create directory command 219
create link command 239
create replica command 223
creating and maintaining accounts 347

creation timestamp 192
credentials 309
cross linking RACF users with DCE users 400, 405, 407
cross linking small numbers of users for RACF interoperability 400
cross-cell authentication accounts 377
cryptography, hardware 465

D

DACL Management interfaces 561
DACL Management, rdaclif 561
daemon
  configuration file 37
  environment variable file 485
daemon, remote procedure call 8
daemons
  availability in OS/390 DCE 17
  DCE, in OS/390 29
  order of starting 35
  starting with DCEKERN 35, 38
  stopping, using MODIFY DCEKERN 36
  viewing the status 36
  ways to start 34
  who can start or stop 34
database 298, 300, 301, 303
database, physical security 300
DCE
  administration
    commands 29
    OS/390 considerations 29
    using the OS/2 GUI 29
  Directory Service
    overview 9
  principals 5
  Threads 8
DCE Audit, auditable events 553
DCE Audit, files 553
DCE Cell Directory Service (CDS) 193
  dcecp commands for managing 193
  object attributes 194
  object types 193
DCE control program, commands for managing ACLs 319
DCE control program, commands for managing DTS 271
DCE control program, control programs 319
DCE control program, creating audit filters 460
DCE control program, enabling and disabling audit logging 463
DCE control program, modifying and querying Audit daemon attributes 463
DCE control program, modifying filters 461
DCE daemon 99
  authorization overview of 149
DCE daemon *(continued)*
  controlling access to the endpoint map 149
  DCE Clients' Access to 150
  endpoint map 118
  endpoint map, controlling access to 149
  giving application servers access to 150
  giving DCE administrators access to 151
  giving unauthenticated users access to 152
  permissions used 149
DCE daemon, starting and stopping 100
DCE daemons, OS/390 29
DCE Directory Service
  introduction 161
  use by DCE components 161
  use by DCE components, RPC 161
  use by DCE Security Service 161
  use by Distributed File Service 161
  use by Distributed Time Service (DTS) 162
DCE Directory Service (CDS) 193
  control programs for managing 193, 195
DCE Directory Service (CDS), control programs for managing 193
DCE names, a closer look at 167
DCE objects, full pathnames 299
DCE principals, definition 5
DCE software clock 289
DCE, default endpoint map ACL 152
DCECONF, using to configure and start 35
dcecp command 271
dcecp command, principal create 334, 335
dcecp commands 203
dcecp commands, account catalog 381
dcecp commands, account create 354, 355
dcecp commands, account delete 355
dcecp commands, account modify 355, 413
dcecp commands, clearinghouse create 248
dcecp commands, clearinghouse delete 250
dcecp commands, clearinghouse disable 247
dcecp commands, group create -uuid 427
dcecp commands, group list 384
dcecp commands, keytab delete 360
dcecp commands, keytab remove 360
dcecp commands, link modify 240
dcecp commands, list 227
dcecp commands, org create -uuid 427
dcecp commands, principal catalog 385
dcecp commands, principal create -uuid 427
dcecp commands, principal modify 335
dcecp commands, properties 417
dcecp commands, registry modify 411, 415
dcecp commands, registry show 415
dcecp commands, required permissions 203
dcecp commands, required permissions for 201
dcecp commands, show 227, 228
dcecp keytab add command 348

dcecp keytab remove command 348
dcecp program 271
dcecp, commands for managing CDS 193, 194
dcecp, description 193
dcecp, permissions for using Security commands 440
dcecp, use with CDS 182
DCEKERN
  address space 33
  restarting 38
  starting the daemons 38
  stopping 34
  using the -nodce option to start 38
default filters 461
default ticket lifetime property 415
defaults Registry Editor command
define cdscache create command 188
defining a cell in an LDAP server 256
Delegation 568
delete directory command 244
delete link command 241
delete permission 200
delete replica command 223
Direct trust relationships 375
directories
  child pointers (CDS) 177
directories, appending errors 235
directories, checking the ACLs for 220
directories, controlling access to 307, 325
directories, creating (CDS) 220
directories, duplicate name problems 235
directories, handling insufficient permissions 235
directories, handling unreachable name failures 235
directories, merging 231, 235
directories, merging errors 235
directories, merging into foreign cells 236
directories, upgrading the cell root directory version 220
directories, upgrading the directory version 220
directory
  child 167
  contents 175, 177
  convergence values defined 225
  creating 219
  deleting 244
  initial hierarchy 219
  management tasks 219
  modifying convergence 225
  modifying replica set 241
  names 167
  reasons for creating 219
  skulking 224
directory convergence set to high 225
directory convergence set to low 226
directory convergence set to medium 225
directory hierarchy
  sample diagram 167
directory information base (DIB) 168
directory information tree (DIT) 168
directory replication, CDS backup procedure 217
directory service
  cell directory service 9
  definition 9
  global directory servers (LDAP and X.500) 10
directory services, cell environment 163
directory synchronize command 224
Directory System Agent (DSA) 168
disable clerk command 209
disable server command 217
disable_time_interval ERA 339
disabling CDS clerks 208
disabling CDS servers 208
distinguished name 168
Distributed File Service (DFS), junction 171
Distributed File Service (DFS), use of CDS 161
Distributed Time Service (DTS)
  maintenance 22
Distributed Time Service (DTS), CDS 162
Distributed Time Service (DTS), creating a new clerk or server 275
Distributed Time Service (DTS), dcecp commands for managing 271
Distributed Time Service (DTS), managing 271, 288
Distributed Time Service (DTS), reconfiguring on nodes 274
Distributed Time Service, introduction 259
DNS (Domain Name System) 12, 170
Domain Name System (DNS) 163, 170
Domain Name System (DNS), name syntax 171
drift 262
DTS
  adjustment of the clock 265
  advantages 260
  applications support 260
  Backus Naur format 539
  basic concepts 262
  computed time 264
  concepts 259
  daemon (dtsd) 13
  features and benefits 259
  management 261
  OS/390 Sysplex environment 290
  overview 12
  relationship with other hosts 13
  relative time format 268
  time conversions 539
DTS control program
  administration tool 13
DTS entity 556

DTS servers, changing names 282
DTS servers, changing the number required 277
DTS servers, checking synchronization with time providers 285, 286
DTS servers, checkinterval attribute 285
DTS servers, designating 283, 285
DTS servers, matching epochs 285
DTS servers, setting response time 282
DTS servers, specific management tasks 283, 286
dts_audit_events 555
dts_audit_events, ACL 555
DTS, access control 287, 288
dtscp commands, clock set 286
dtsd (DTS daemon) 13

E

EBCDIC, conversion to ASCII 181
editing an ACL associated with CDS clerk or server 201
editing an initial container creation ACL 198
editing an initial object creation ACL 198
endpoint map
  and DCED 118
  commands for monitoring 119
  commands for rebuilding 119
  controlling access to 149
  deleting during startup 120
  overview 8
  recovering 120
  recovering a corrupted 121
  recovering from I/O errors 121
endpoint maps 118
endpoints, purging obsolete 118
entities 181
entity 182
envar file 483
ENVAR runtime option 484
environment variable file 483
environment variable files
  of DCE daemons 485
environment variable files of DCE daemons 485
environment variables
  _EUV_ENVAR_FILE 483
  description 467
  envar file 483
  environment variable file 483
  file 467
  for client and server controls 468
  for DCECONF controls 480
  for dcecp Controls 467
  for debug controls 474
  for GDA daemon controls 468
  for IDL compiler controls 482
  for message controls 473
  for messaging subsystem 485
environment variables *(continued)*
  for OS/390 kernel controls 472
  for security server controls 471
  for server controls 471
  for single sign-on controls 474
  format for setting 483
  how to set 483
  Kerberos environment variables 482
  POSIX variables used by DCE 467
  RPC_DEFAULT_ENTRY 123
  setting from batch or TSO 484
  setting from the shell 484
epoch 269
epoch number 269
epochs, matching 285
ERA facility 297
ERA, disable_time_interval 339
ERA, max_invalid_attempts 339
ERA, pre_auth_req 337
ERA, pwd_mgmt_binding 339
ERA, pwd_val_type 339
ERAs, deleting 440
ERAs, handling logins that are not valid 336
ERAs, modifying 441
ERAs, password management 336
ERAs, permissions to create 440
ERAs, preauthentication 336
ERAs, security 336
ERAs, viewing 441
error tolerance characteristic
ESM, using instead of RACF 389
event class 446
event class file 447
event class name 447
event class number 447
event class, auditing execution 555
event class, defining 459
event class, definitions 553, 555, 560
event number 446
events, audit service operations 553
events, audit services 553
events, auditable 553, 555, 559
events, clock readings 555
events, dts_audit_events 555
events, global-set membership 555
events, security service operations 560
events, security services 559
events, time service attributes 555
events, time service processes 555
events, time services 555
expired user account 356
extended ACL entry type 315
Extended Registry Attributes 363
Extended Registry Attributes (ERA 568
extracell naming 163

F

faulty clocks, detecting 264
file name, global 172
file system full condition 212
files, aud_audit_events 553
files, controlling access to 307, 325
files, dts_audit_events 553
files, sec_audit_events 553
files, security administration 553
filter 448
filter guides 449
filter, creating and maintaining 460
filter, subject identity 449
filters, default 461
foreign cell, creating an account for 23
foreign_group entry type 312, 313
foreign_other entry type 312, 313
foreign_user entry type 313
full names 331
full names, changing 438
functions, rdacl_get_access() 561
functions, rdacl_get_manager_types() 562
functions, rdacl_get_referral() 562
functions, rdacl_lookup() 561
functions, rdacl_replace() 561
functions, rdacl_test_access() 561
functions, rpriv_get_ptgt() 562
functions, rs_acct_add() 563
functions, rs_acct_delete() 563
functions, rs_acct_get_projlist() 563
functions, rs_acct_lookup() 563
functions, rs_acct_replace() 563
functions, rs_auth_policy_get_effective() 567
functions, rs_auth_policy_get_info() 567
functions, rs_auth_policy_set_info() 567
functions, rs_login_get_info() 564
functions, rs_pgo_add_member() 565
functions, rs_pgo_add() 564
functions, rs_pgo_delete_member() 565
functions, rs_pgo_delete() 564
functions, rs_pgo_get_members() 566
functions, rs_pgo_get() 565
functions, rs_pgo_is_member() 565
functions, rs_pgo_key_transfer() 565
functions, rs_pgo_rename() 564
functions, rs_pgo_replace() 564
functions, rs_policy_get_info() 566
functions, rs_policy_set_info() 566
functions, rs_properties_get_info() 566
functions, rs_properties_set_info() 566
functions, rs_rep_admin_maint() 567
functions, rs_rep_admin_mkey() 567
functions, rsec_id_gen_name() 568
functions, rsec_id_parse_name() 568
functions, rsec_krb5rpc_sendto_kdc() 560

G

gbl_time_service, Time Server 555
GDA (global directory agent), overview 12
gdad command 254
gdad process 254
global DCE file name 171
global directory agent (GDA)
  description 164
  how CDS finds 251
  how it works 251
Global Directory Agent (GDA), how it works 254
Global Directory Agent (GDA), managing 254
global file name 172
global name 165
global namespace 5
global OS/390 DCE files 41
global principal name 171
global server set 270
global servers, advertising 283, 284
global servers, changing required 278
globaltimeout attribute 282
granting permissions, effect on ACL checking sequence 319
group directory 429
group entry type 312, 313
group IDs, setting in registry 416
group_obj entry type 311, 312
groups
  changing 343
  definition 298
  deleting 343
  example, adding 342
  permissions required for
    adding members 437
    changing information 438
    creating 432
    deleting 433
    deleting members 437
  project lists 341
groups, accrual of permissions 314
groups, ACL entry types 311
groups, adding members 437
groups, adding to organization 342
groups, adding to registry 432
groups, changing full names 438
groups, changing management information 438
groups, changing registry information 343
groups, creating and maintaining 331
groups, deleting 343, 433
groups, deleting members 437, 438
groups, displaying registry information 382, 384
groups, excluding from project lists 314

groups, logical identities 299
groups, management information 431
groups, membership lists 344
groups, naming restrictions 331

H
HFS to PDS, mapping of 42
hidden password property 415
Host Services 99
Host Services, starting and stopping 100

I
I/O error during usual operation, recovering from 121
identifier Mapping interfaces 568
Identifier Mapping, secidmap 568
identity 347
improving registry performance 24
inaccuracy 261, 262
inaccuracy values, example 274
information, where to find more xxx
Initial Container ACL 321, 325
Initial Container Creation ACL 198
initial container creation ACL, editing 198
Initial Object ACL 321, 325
Initial Object Creation ACL 197
initial object creation ACL, editing 198
insert permission 200
intercell naming 163, 251
interface, RPC identifier 130
interfaces, network, used by DCE 20
interim file 231
interoperability, RACF and single sign-on 389
interprocess communication 268
interval 259, 262
interval, non-intersecting 264
ISPF
  entering arguments on command line 30

J
junction 171

K
kdestroy command 351
key entries, deleting from keytab file 360
key tables, displaying registry information 387
keys, deleting from keytab file 358
keys, machine and server accounts 361
keys, version numbers 358
keytab file 348
keytab file, adding keys 358
keytab file, dced object 357
keytab file, protecting 357
keytab files, deleting 360
klist command 350
krb5rpc interface 560
krbgt directory 375
krbtgt directory, in multicell environment 375

L
LDAP server, defining a cell in 256
leap seconds 262
local identity 347
local names 166
local OS/390 DCEfiles 41
local server set, DTS 269
localtimeout attribute 282
log file 303
login 336
login activity 356
login, "handling" 339
lookup
  definition 173
  diagram 174
  how CDS finds names 188
  role of child pointer 188, 189

M
maintenance tasks
  CDS 20
  Security Service 23
maintenance, overview 19
management information 430
mask 315
mask_obj entry type, effect on ACL checking 318
mask_obj mask 315, 318
mask, unauthenticated 319
masks, types and use of 315
master keys, backing up 420
master keys, changing 419
master keys, restoring 421
master registry database 14
master replica 175, 302
math server
  determining frequency of use 139
  example 139
  RPC profile, steps in creating 140
  solution 140
max_invalid_attempts ERA 339
maximum ticket lifetime policy 413
maximum ticket lifetime property 416
maximum ticket renewable time policy 413
maximum UNIX ID property 416
maxinaccuracy attribute 279, 280
members 344

membership lists 344
merging CDS directories, overview 231
merging, overview of procedure 231
messaging subsystem environment variables 485
minimum restart interval, daemon configuration file 37
minimum ticket lifetime, setting in registry 416
minservers attribute, changing 277
MODIFY DCEKERN operator command 35
MODIFY DCEKERN, to start daemons 35
monitoring HFS utilization 39
monotonicity 259
multi-cell considerations for RACF interoperability 398
multicell, administering 375
mvsexpt 390
mvsexpt, tailoring variables 394
mvsimpt 390
mvsimpt and mvsexpt, tailoring variables 393
mvsimpt, tailoring variables 393

N

name service interface (NSI) 176
names 5
- alias 176
- cell-relative 166
- changing full names 438
- comparison of CDS and GDS 169, 170
- distinguished 168
- DNS 166
- full 167
- GDS 168
- global 165
- how CDS looks up 174, 188, 190
- how CDS protects 180
- in standalone cell 166
- local 166
- maximum sizes 533
- naming rules 529
- organizing 167
- primary name, registry object 331
- principal, group and organization 331
- principal, group, organization 331
- relative distinguished 168
- searching for 170
- simple 167
- valid characters 529

namespace
- authorization group, adding members 204
- cell 167
- definition 5
- global 5
- logical and physical views 183
- preferred clearinghouse for viewing 229
- restructuring 176, 239

namespace *(continued)*
- viewing the content 227
- viewing the structure 227

namespaces, managing 21
naming
- extracell 163
- intercell 163, 251

network identity 347
Network Information Center (NIC) 170
network interfaces
- restricting those used by DCE 20

NIC (Network Information Center) 170
NIST (National Institute of Standards and Technology) 259
nodce option, starting DCEKERN 38
notices xxvii
NSI (Name Service Interface)
- maintenance 19

null time provider 289
null time provider program, definition 289
number formats in registry 333

O

object 176
Object ACL 197, 321, 325
object attribute 9
object creation quotas 333
object entry
- clearinghouse 176
- clearinghouse, diagram 184
- creating 176

object identifiers, origin of 535
objects, ACL editing 321
objects, controlling access to 307, 325
objects, default ACLs 321
objects, initial access control lists 321
objects, quotas for creating 378
objects, types of 321
operator command, MODIFY DCEKERN 35
org directory 429
organization IDs, setting in registry 416
organization name 352
organizations
- adding 342
- definition 298
- deleting 343
- permissions required for
  - adding members 437
  - changing management information 438
  - creating organizations 432
  - deleting members 437
  - deleting organizations 433

organizations, adding members 437
organizations, adding to registry 342, 432

organizations, changing full names 438
organizations, changing management information 438
organizations, changing registry information 343
organizations, creating and maintaining 331
organizations, deleting 343, 433
organizations, deleting members 437, 438
organizations, displaying registry information 382, 384
organizations, logical identities 299
organizations, management information 431
organizations, membership lists 344
organizations, naming restrictions 331
organizations, policies 415
orphans, adopting 333, 427, 428
OS/390
- DCE administration considerations 29
- DCE daemons 29

OS/390 DCE
- environment variables 467
- product architecture 8
- starting and stopping 33
- starting during system IPL 37
- using DCECONF to start 35

OS/390 Security Server interfaces 560
other_obj entry type 311, 312

P

Parallel Sysplex
- workload balancing with 153

parent directory 167
password management server, configuring 340
password, managing generation of 339
password, managing generation of pwd_strengthd 339
password, managing strength of 339
password, managing strength of pwd_strengthd 339
passwords 336
- expiration date policy 412
- format policy 412
- lifespan policy 412

passwords, changing 439
passwords, changing in cross-cell authorization accounts 379
passwords, effects of policy changes 414
passwords, encrypting 415
passwords, expiration date 412
passwords, expiration information 350
passwords, format 412
passwords, lifespan 412
passwords, permissions 439
passwords, server accounts 348
pathnames for Security objects 304
pathnames, for registry objects 304
peer-to-peer relationships 376
performance, improving registry 24
permission granting, checking order 319
permission, control 180
permissions
- ACL, definition 307
- CDS 180, 200
- groups and project lists 314
- propagation 198
- registry objects 430
- used by the DCE daemon 149
- valid for ACL managers 442

permissions for cdscp commands 203
permissions for dcecp commands 201, 203
permissions, accrual 314
permissions, accrual, effect on ACL checking 318
permissions, ACL entry syntax 311
permissions, checking sequence 316, 319
permissions, denying 319
permissions, inheriting 309
permissions, principals and groups 311
permissions, registry objects 430, 442
permissions, restricting 315
permissions, scope of UNIX permission bits 308
permissions, setting default 321, 325
permissions, setuid bit 309
PI program for workload balancing 155
policies, authentication 413
policies, authentication information 432
policies, changing 439, 440
policies, handling conflicting 414
policies, management information 430, 431
policies, standard 411, 415
policy
- changes on standard, effects of 414
- conflicting, handling 414
- maintaining 411
- overrides 23
- setting 411
- setting and maintaining 23
- standard 411
- stricter standard 414
- types of 411

POSIX time format 291
POSIX, scope of permissions and DCE ACLs 308
pre_auth_req ERA 337
preauthentication 336
preauthentication, interoperability 338
preauthentication, pre_auth_req ERA 337
preauthentication, third-party 336
preauthentication, timestamps 336
preparing for DCE single sign-on 409
primary name 331, 345
primary names, changing 345

primary names, format 332
principal directory 429
principal name, global 171
principals
  adding, example 432
  authentication 307
  DCE 5
  definition 298
  deleting 336
  permissions to delete 433, 438
  Security service 14
principals, accounts for foreign 375, 379
principals, accrual of group permissions 314
principals, ACL entry types 311
principals, adding to registry 334, 335, 432
principals, authenticating 349, 351
principals, authentication information 432
principals, changing full names 438
principals, changing management information 438
principals, changing registry information 335
principals, creating and maintaining 331
principals, DCE 5
principals, deleting 433
principals, displaying registry information 384, 385
principals, logical identities 299
principals, management information 431
principals, name format 332
principals, naming restrictions 331
principals, network and local identities 347
principals, reserved 332, 333
print server
  example 142
  frequency of use 143
  RPC profile, steps in creating 143
  solution 143
privilege attribute 309
privilege attributes, about 349
privilege attributes, viewing 350
Privilege Server interfaces 562
Privilege Server, rpriv 562
Privilege Service 14
process 309
Processed Entries file, recreating 400
profile
  cell 270
  LAN 269
profile, cell 284
project lists 341, 342, 343
project lists, definition 314
project lists, excluding groups from 314
project lists, in ACL checking sequence 316
propagation 304
propagation queue 303
propagation queues 303
properties 411, 415
properties, authentication information 432
properties, changing 439, 440
properties, management information 430, 431
properties, setting 415, 417
protocol, solicitation and advertisement 188
pwd_mgmt_binding ERA 339
pwd_val_type ERA 339

Q

queryattempts attribute 282
quotas, for creating registry objects 378

R

RACF interoperability and single sign-on 15, 389
RACF interoperability utilities 390
RACF interoperability, cross linking a few users 400
RACF interoperability, users in different cells 398
RACF-DCE interoperability, overview 389
rdacl_get_access() function 561
rdacl_get_manager_types() function 562
rdacl_get_referral() function 562
rdacl_lookup() function 561
rdacl_replace() function 561
rdacl_test_access() function 561
rdaclif interface 560
rdaclif operations 561
read permission 200
recovering from a corrupted CDS cache 211
recovering from cdsd file system full condition 212
recreating the Processed Entries file 400
redirection, command input and output 32
registries, accessing objects 429, 444
registries, account lifespan 411
registries, authentication policies 413
registries, changing the master replica site 423, 424
registries, default ticket lifetime property 415
registries, deleting when machines are removed 424
registries, displaying and setting policies 415
registries, hidden password property 415
registries, maintaining policies and properties 411, 417
registries, maximum ticket lifetime 413
registries, maximum ticket renewable time 413
registries, maximum UNIX ID property 416
registries, minimum group ID property 416
registries, minimum organization ID property 416
registries, minimum ticket lifetime property 416
registries, minimum UNIX ID property 416
registries, password expiration date 412
registries, password format 412

registries, password lifespan 412
registries, policies 411
registries, registry-wide, policies 415
registry
- database object 441
- initial ACLs 443
- master registry database 14
- permissions 430
- policies 439
- policy 411
- using the editor 23

Registry ACL Manager 442
Registry Administration 567
Registry Administration interfaces 567
registry connect command 23, 376
registry database, backing up 420
registry database, changing master key 419
registry database, range for UNIX numbers 333
registry database, routine maintenance 419
registry database, storing 300
registry database, structure and contents 429, 430
registry database, updating 301
registry database, viewing information 381
Registry Editor
- and the Security service 14
- creating and maintaining accounts 23
- Security service maintenance tasks 23

Registry miscellaneous operations interfaces 564
Registry miscellaneous operations, rs_misc 564
registry objects, accessing 429
registry objects, adopting 333
registry objects, adopting orphans 427, 428
registry objects, full pathnames 300
registry objects, pathnames 304
registry objects, permissions 430, 442
registry orphans, adopting 427
registry performance, improving 24
Registry PGO interfaces 564
Registry PGO, rs_pgo 564
Registry Policy interfaces 566
Registry Policy, rs_policy 566
registry properties 439
Registry Server Attributes 568
Registry Server interfaces 563
Registry Server, rs_acct 563
Registry Service 14
Registry, extending 363
registry, initial object ACLs 443, 444
registry, reconfiguring 24
registry, table of access permissions to objects 430
relative distinguished name 168
remote procedure call (RPC)
- cell 5
- definition 4, 5
- overview 8

remote procedure call (RPC), definition 4
Remote Procedure Call (RPC), use of CDS 161
remote procedure call daemon 8
replica
- CDS directories 175
- contents 175, 177
- creating 223
- definition 301
- deleting 223
- designating new master 242
- maintaining consistency 191, 192
- pre-creation tasks 221
- read-only 221
- reasons for creating 221
- reasons for deleting 223
- replica set
  - excluding replica 243
  - modifying 241
- types 175

replica list 304
replica lists 304
replicas 303
replicas, creating (CDS) 223
replicas, deleting slaves (Security) 424
replicas, restarting master (Security) 303
replicas, slave permissions (Security) 442
replicas, updating (Security) 304
replication
- definition 6
- purpose 6
- registry database 14

resource record, cell 255
restarting automatically after ABEND 210
restarting the CDS advertiser and CDS clerk 209
restricting endpoints 121
restricting network interfaces used by DCE 20
rgy_edit commands, cell 379
RPC
- endpoint map 19
- endpoint map, controlling access 19
- NSI (Name Service Interface), maintenance 19
- profile
  - examples of setting up 139
- server group 150

RPC ACL manager, ACL entry types supported by the 149
RPC control program
- commands for monitoring the endpoint map 119
- commands for rebuilding the endpoint map 119
- ping command 120

RPC server group 150
rpc_c_authn_dce_secret authentication service 560
RPC_DEFAULT_ENTRY environment variable 123
rpc_ep_register routine 118

rpc_ep_register_no_replace routine 118
RPC_RESTRICTED_PORTS 121
RPC, interface identifier 130
rpcd (remote procedure call daemon) 8
RPCD, definition
rpriv interface 560
rpriv operations 562
rpriv_get_ptgt() function 562
rs_acct interface 560
rs_acct operations 563
rs_acct_add() function 563
rs_acct_delete() function 563
rs_acct_get_projlist() function 563
rs_acct_lookup() function 563
rs_acct_replace() function 563
rs_auth_policy_get_effective() function 567
rs_auth_policy_get_info() function 567
rs_auth_policy_set_info() function 567
rs_login_get_info() function 564
rs_misc operations 564
rs_pgo operations 564
rs_pgo_add_member() function 565
rs_pgo_add() function 564
rs_pgo_delete_member() function 565
rs_pgo_delete() function 564
rs_pgo_get_members() function 566
rs_pgo_get() function 565
rs_pgo_is_member() function 565
rs_pgo_key_transfer() function 565
rs_pgo_rename() function 564
rs_pgo_replace() function 564
rs_policy operations 566
rs_policy_get_info() function 566
rs_policy_set_info() function 566
rs_properties_get_info() function 566
rs_properties_set_info() function 566
rs_query interface 560
rs_rep_admin_maint() function 567
rs_rep_admin_mkey() function 567
rs_rpladmn interface 560
rs_update interface 560
rsec_cert interface 560
rsec_id_gen_name() function 568
rsec_id_parse_name() function 568
rsec_krb5rpc_sendto_kdc() function 560
rules, BNF 539

S

schema 168, 363
schema entries, displaying 385
sec_admin command, description 330
sec_admin commands, for reconfiguring replica sets 330
sec_admin, permissions for using commands 440
sec_audit_events 559
sec/group directory 512
sec/group/subsys directory 516
sec/principal directory 520
sec/principal/hosts directory 525
secidmap interface 560
secidmap operations 568
security 180
security administration, auditable events 553
security administration, files 553
security administration, introduction 553
security client 297
security database 297
security directory, top-level 509
security group, DTS server 284
security namespace 509
security server 297
Security Service
  ACL Facility 14
  Authentication Service 14
  CDS 161
  CDS junction 171
  Login Facility 14
  maintenance tasks 23
  overview 14
  principals 14
  Privilege Service 14
  Registry Service 14
Security Service commands, kdestroy 351
Security Service commands, klist 350
Security Service principals 14
Security Service, dcecp commands for managing 328
security service, overview 297
security services, auditable events 559
security, ERAs 336
security, passwords 336
server
  CDS 167, 173
  controlling access 200
  counters 207
  courier 270
  diagram 177
  disabling 209
  DTS 259, 269
  example 3
  gaining access to namespace 204
  global
  list 268
  local 269
  management
  management tasks 207
  monitoring counters 207
server accounts 347
server accounts, passwords for 348

server is listening, commands for determining if 120
server machines, removing from network (Security) 423
server system upgrade, preserving clearinghouse 211
server, advertising outside a cell 284
server, global, accessing outside a cell 284
server, password management, configuring 340
serverentry attribute 282
serverprincipal attribute 282
servers, managing 21
service tickets 350
set directory command 226
set directory to new epoch command 241, 242, 243
set link command 240
setting up workload balancing 154
setuid 309
show cell command 254
show clearinghouse command 207
show clerk command 207
show directory command 242
show server command 207
simple name 167
single sign-on for OS/390 and DCE 409
single sign-on to DCE, automatic 410
single sign-on to DCE, preparing for 409
skew 262
skulk operation 191, 192
skulk time, background 192
skulk, reasons for initiating 224
skulks 20
slave replica 302
slave replicas 14
soft link
  creating 239
  deleting 241
  description 176
  management 239
  modifying destination name 240
  modifying expiration time 240
  recommended usage 239
  sample resolution 185
soft link modifying extension time 240
soft links, changing destination names 240
soft links, creating 239
soft links, expiration and extension values 240
soft links, with timeout attribute 239
software clock, DCE 289
software clock, resetting 291
solicitation and advertisement protocol 188
stale cache entries, removing 210
standard policies 411
starting daemons with DCEKERN 35
Starting DCE host services 100
starting the CDS daemon 209
Stopping DCE host services 100
stopping the CDS daemon 209
subtrees, merging 231
synchronization process 263
synchronization process, computation of correct time 263
synchronize 263
synchronize command
synchronizing CDS server clocks 225
syncinterval attribute 280
Sysplex timer 290
system administration, parts 17
system time
  modifying the 23
system time, adjusting clocks 286
system time, changing 286, 287
system time, changing monotonically 286
system time, setting abruptly 286
system time, setting gradually 286
system time, updating nonmonotonically 286

T

tailoring the RACF utilities 393
tasks, maintenance CDS 20
tasks, Security Service 23
TDF (time differential factor) 267
test permission 180, 200
threads 7
ticket cache name 351
ticket-granting ticket 350
tickets, about 350
tickets, deleted passwords 358
tickets, destroying 351
tickets, expiration and renewal 350
tickets, setting lifetimes 353, 354, 416
tickets, status when process halts 351
tickets, ticket lifetime 413
tickets, ticket-granting ticket validity 413
tickets, viewing information 351
time
  absolute
    display format 266
    representation 266
  computed 263
  displays 265
  inaccuracy 263
  local translation 267
  relative 267
  relative, representation 267
  representation 265
time differential factor (TDF) 267
time inaccuracy values, maximum 279, 280
time interval 259

time measurement factors 262
time offset, DCE software clock 289
time out period, daemon configuration file 37
Time Provider, interfaces 559
time providers 259, 260
time providers, checking server synchronization 285, 286
time providers, support for 260
Time Server, clock readings 555
Time Server, gbl_time_service 555
Time Server, global-set membership 555
Time Server, Time Service 555
Time Server, time service attributes 555
Time Server, time service processes 555
Time Server, time_control 555
Time Server, time_provider 555
Time Server, time_service 555
Time Service, Even-Specific Information 556
Time Service, Event Classes 556
Time Service, Event Types 556
Time Service, interfaces 556
Time Service, Time Server 555
time services, auditable events 555
time zone, overriding the system 291
time zone, setting 290
time_control, Time Server 555
time_provider, Time Server 555
time_service, Time Server 555
timestamp
  binary 265
  CDS use 192
  creation 192
  displays
  update 192
timestamps, format 273, 274
tolerance attribute 281, 282
TPI (time-provider interface) 259
trademarks xxvii
traditional backups, CDS backup procedure 217
trail file 451
Transitive trust relationships 375
translating from names to resources 183
trust relationships 375
TZ, using POSIX time format 291

U

UID (unique identifier) 333
unauthenticated mask 315
unauthenticated mask, effect on ACL checking 319
unauthenticated mask, inappropriate use of 319
unique identifier 427
unique identifier (UID) 333
UNIX IDs, for foreign users 376
UNIX IDs, setting in registry 416
UNIX numbers 333
update 259
update propagation 191
update timestamp 192
user account expiration 356
user accounts 347
user entry type 311, 313
user information 432
user interfaces, CDS 181
user_obj entry type 311, 312
user, commands 29
using the Control Task to monitor HFS utilization 39
using workload balancing 154
UTC (Coordinated Universal Time) 259, 265
UTC, leap seconds 262
utilities for backing up files 42
UUIDs and UNIX IDs 333

W

workload balancing
  client requirements 154
  in a Parallel Sysplex environment 153
  server requirements 154
  setting up 154
  using 154
  using the APIs 155
  using the dcecp endpoint commands 155
  using the environment variable 154
  using the PI program 155
write permission 180, 200

X

xattrschema object 363
xattrschema, displaying registry information 385

Readers' Comments

OS/390®
DCE
Administration Guide

Publication No. SC28-1584-05

You may use this form to report errors, to suggest improvements, or to express your opinion on the appearance, organization, or completeness of this book.

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note

Report system problems to your IBM representative or the IBM branch office serving you.
U.S. customers can order publications by calling the IBM Software Manufacturing Solutions at **1-800-879-2755**.

In addition to using this postage-paid form, you may send your comments by:

FAX	1-607-752-2327	Internet	pubrcf@vnet.ibm.com
IBM Mail	USIB2L8Z@IBMMAIL	IBMLink	GDLVME(PUBRCF)

Would you like a reply? ___**YES** ___ **NO** If yes, please tell us the type of response you prefer.

___ Electronic address: ______________________________

___ FAX number: ______________________

___ Mail: (Please fill in your name and address below.)

Name

Address

Company or Organization

Phone No.

Readers' Comments
SC28-1584-05

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Department G60
International Business Machines Corporation
Information Development
1701 North Street
ENDICOTT NY 13760-5553

Fold and Tape **Please do not staple** Fold and Tape

SC28-1584-05

Cut or Fold
Along Line

Program Number: 5647-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC28-1584-05

Spine information:

OS/390 DCE Administration Guide