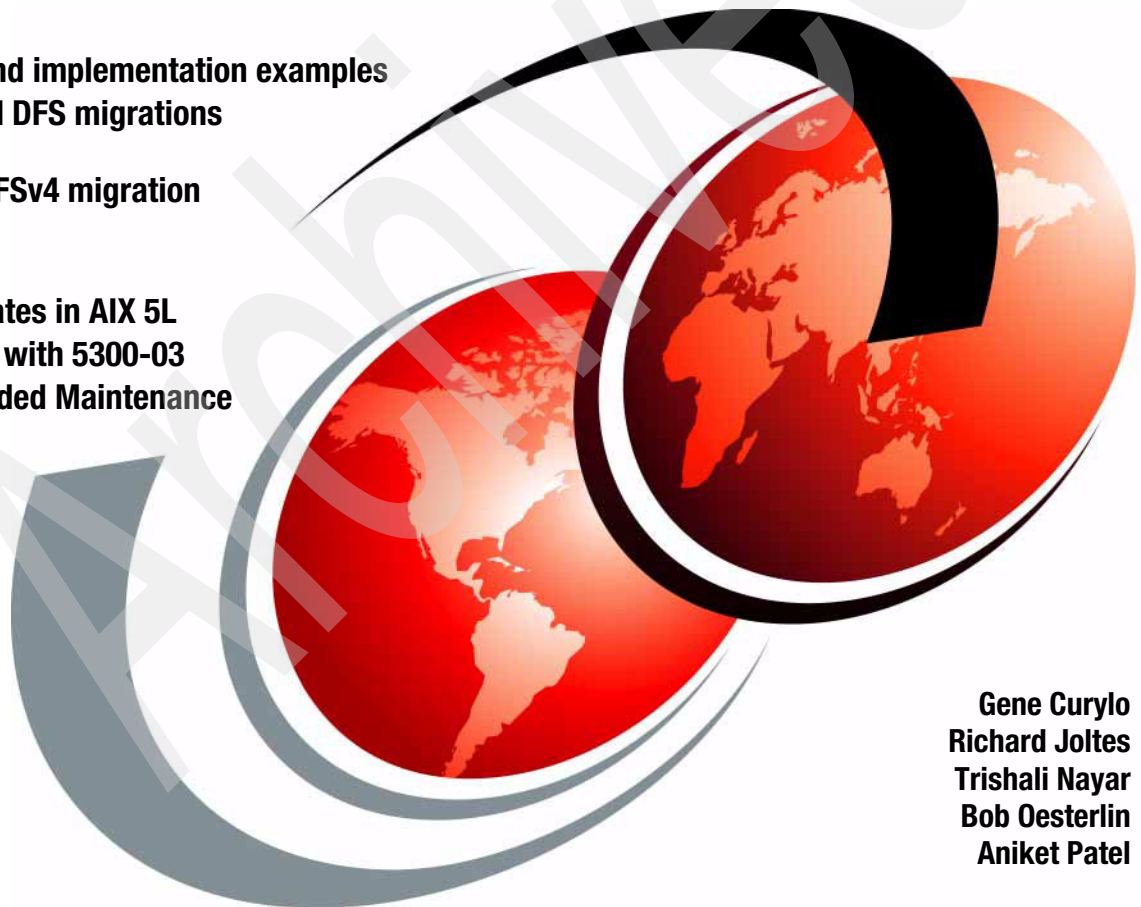


Implementing NFSv4 in the Enterprise: Planning and Migration Strategies

Planning and implementation examples
for AFS and DFS migrations

NFSv3 to NFSv4 migration
examples

NFSv4 updates in AIX 5L
Version 5.3 with 5300-03
Recommended Maintenance
Package



Gene Curylo
Richard Joltes
Trishali Nayar
Bob Oesterlin
Aniket Patel



International Technical Support Organization

Implementing NFSv4 in the Enterprise: Planning and Migration Strategies

December 2005

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (December 2005)

This edition applies to Version 5, Release 3, of IBM AIX 5L (product number 5765-G03).

© Copyright International Business Machines Corporation 2005. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Noticesxi
Trademarks	xii
Preface	xiii
The team that wrote this redbook	xiv
Acknowledgments	xv
Become a published author	xvi
Comments welcome	xvii
Part 1. Introduction	1
Chapter 1. Introduction	3
1.1 Overview of enterprise file systems	4
1.2 The migration landscape today	5
1.3 Strategic and business context	6
1.4 Why NFSv4?	7
1.5 The rest of this book	8
Chapter 2. Shared file system concepts and history	11
2.1 Characteristics of enterprise file systems	12
2.1.1 Replication	12
2.1.2 Migration	12
2.1.3 Federated namespace	13
2.1.4 Caching	13
2.2 Enterprise file system technologies	13
2.2.1 Sun Network File System (NFS)	13
2.2.2 Andrew File System (AFS)	14
2.2.3 Distributed Computing Environment/Distributed File System	15
2.3 General considerations when using enterprise file systems	16
Part 2. NFSv4 on AIX 5L V5.3	19
Chapter 3. NFSv4 implementation	21
3.1 Implementation of the NFSv4 protocol in AIX 5L V5.3	22
3.2 NFSv4 features supported in the initial AIX 5L V5.3 release	22
3.2.1 External namespace (exname) support	23
3.2.2 FSIDs and file handles	25
3.3 Features introduced in AIX 5L V5.3 RML03	26
3.3.1 Delegation	27

3.3.2 Referral	30
3.3.3 Replication	36
3.4 List of NFSv4 features supported in AIX 5L V5.3	50
Chapter 4. Using NFSv4 with JFS2 or GPFS	53
4.1 AIX 5L enhanced journaled file system (JFS2)	54
4.1.1 Comparing JFS2 with JFS	56
4.1.2 JFS2 advanced features	57
4.1.3 Using JFS2 with NFSv4	59
4.1.4 JFS2 ACLs versus NFSv4 ACLs.	59
4.1.5 How do we implement inheritance NFSv4 ACLs?	62
4.2 General Parallel File System (GPFS)	63
4.2.1 Why GPFS?	66
4.2.2 GPFS advantages.	67
4.2.3 When to consider GPFS	70
4.2.4 Planning considerations for GPFS	70
4.2.5 Using NFSv4 with GPFS.	70
4.2.6 NFSv4 export considerations for GPFS	72
4.2.7 NFS usage of GPFS cache.	72
4.2.8 NFSv4 ACL administration	72
4.2.9 NFS client with stale inode data	79
4.3 Backup considerations	79
Chapter 5. Using NFSv4 features.	81
5.1 Using the cache file system (CacheFS)	82
5.1.1 CacheFS performance benefits	82
5.1.2 CacheFS performance impacts.	83
5.1.3 Configuring CacheFS	84
5.2 Managing LDAP automount maps	85
5.3 Pseudo file system	87
5.4 NFSv4 ACLs	90
5.4.1 NFSv4 ACLs: ACL evaluation.	95
5.4.2 NFSv4 ACLs: Administration.	98
5.4.3 NFSv4 ACLs: ACL inheritance and umask.	103
5.4.4 NFSv4 ACLs: Permissions scenarios	110
5.4.5 NFSv4 ACLs: ACL evaluation flowchart for NFSv4	112
5.4.6 NFSv4 ACLs: NFSv3 clients	114
Part 3. Preparing to use NFSv4	115
Chapter 6. Building an NFSv4 environment	119
6.1 Environment used for demonstration scenarios	120
6.2 Infrastructure setup flow	120
6.3 Network Time Protocol (NTP) configuration	122

6.4 IBM Tivoli Directory Server V5.2	122
6.4.1 Preparing the system for IBM Tivoli Directory Server installation . .	124
6.4.2 Installing IBM Tivoli Directory Server	126
6.4.3 Configuring IBM Tivoli Directory Server	127
6.4.4 Configuring Tivoli Directory Server to be a client of itself	133
6.5 IBM Network Authentication Services (Kerberos V5) server installation .	134
6.5.1 Setting up the environment	134
6.5.2 Configuring the NAS server	134
6.6 IBM Tivoli Directory Server client configuration	140
6.7 IBM Network Authentication Services client install and configuration . .	141
6.7.1 Integrated login (single sign-on)	141
6.7.2 Standard login	143
6.7.3 Adding NAS users	144
6.7.4 Migrating existing users into NAS	147
6.7.5 Installation details	149
6.8 Installing GPFS	150
6.8.1 Preparing the GPFS nodes for installation	151
6.8.2 Creating the GPFS directory	151
6.8.3 Creating the GPFS installation table of contents file	152
6.8.4 Installing GPFS through the network	152
6.8.5 Verifying the GPFS installation	153
6.9 Configuring GPFS	153
6.9.1 Setting up the environment	154
6.9.2 Creating the GPFS cluster and nodes	155
6.9.3 Creating a GPFS file system	156
Chapter 7. Migration considerations	161
7.1 General migration considerations	163
7.2 Types of migrations	164
7.2.1 Switch-over migration	164
7.2.2 Phased or rolling migration	165
7.2.3 User-by-user or self-managed migration	166
7.3 Hardware planning	168
7.4 Individual component considerations	169
7.4.1 Security	169
7.4.2 RPCSEC_GSS security flavors	170
7.4.3 RPCSEC_GSS protection levels	170
7.4.4 User identity management options	171
7.4.5 User and group identities and NFSv4	172
7.4.6 RPCSEC_GSS user authentication using Kerberos	172
7.4.7 User accounts and authentication resources	173
7.5 NFSv4 user authorization methods	178
7.5.1 Choosing a user authorization method	178

7.5.2 Other user authorization considerations	179
7.5.3 NFSv4 host identification	181
7.5.4 NFSv4 host authentication	181
7.5.5 NFSv4 host authorization	182
7.6 Choosing the appropriate file system types	183
7.6.1 Backup systems	183
7.6.2 Time services	184
7.6.3 User data.	185
Chapter 8. Migration scenarios	189
Part 4. Migrating to NFSv4	193
Chapter 9. NFSv3 to NFSv4 migration.	195
9.1 The test environment.	197
9.2 Using NFSv3 and NFSv4 side-by-side	198
9.3 Migrating from NFSv3 to NFSv4	199
9.4 Using NFSv3	201
9.5 Using NFSv4 with NFSv3	202
9.5.1 Configuring the NFS domain.	202
9.5.2 Configuring the pseudo root file system	203
9.5.3 Exporting file systems for access to NFSv3 and NFSv4 clients	204
9.5.4 Mounting NFSv4 exports on the clients	204
9.5.5 Mounting NFSv3 exports on the clients	205
9.5.6 Differences between NFSv3 and NFSv4 mounts	206
9.6 Adding security	206
9.6.1 Creating NFS service principals in Kerberos	207
9.6.2 Configuring the gssd daemon on the NFS server	208
9.6.3 Mapping Kerberos V5 realms to NFS domains.	208
9.6.4 Creating the NFS keytab file entry	209
9.6.5 Configuring security on the clients.	210
9.6.6 Exporting NFS file systems with security	213
9.6.7 Mounting an NFSv4 exported file system	214
9.7 Namespace management	214
9.7.1 How does the NFSv4 namespace help?	216
9.7.2 Enhancing classic NFSv4 exports using the exname option	217
9.8 Setting a different pseudo root file system	220
Chapter 10. Planning a migration from DFS	223
10.1 An overview of DCE/DFS	224
10.1.1 Servers and clients	224
10.1.2 Cells	225
10.1.3 Cross-cell communications	226
10.1.4 Caching	226

10.1.5	Aggregates and filesets	227
10.1.6	Replication	227
10.2	Component-specific migration considerations	228
10.2.1	Authentication services	228
10.2.2	DCE/DFS principal and group considerations	229
10.2.3	Migrating accounts from DCE to Kerberos V5	230
10.2.4	Authentication methods	235
10.2.5	Additional considerations	235
10.3	ACL migration considerations	237
10.3.1	Understanding DFS ACL evaluations	237
10.3.2	DFS to NFSv4 ACL translation	240
10.3.3	DFS and NFSv4 ACL comparisons	240
10.3.4	Example of DFS to NFSv4 ACL translation	240
10.3.5	Data migration	243
Chapter 11.	Illustrated DFS migration	247
11.1	Test environment	249
11.2	Migrating the DCE cell to LDAP/KRB5	249
11.3	Migrating user data	256
11.3.1	Capturing existing ACLs in the DFS environment	256
11.3.2	Copying data from DFS to the NFS namespace	257
11.3.3	Restoring ACLs on the copied data	258
Chapter 12.	Planning a migration from AFS	261
12.1	A broad overview of AFS	262
12.1.1	A distributed file system	262
12.1.2	Servers and clients	262
12.1.3	Cells	262
12.1.4	Transparent access and the uniform namespace	264
12.1.5	Security: Mutual authentication and access control lists	264
12.1.6	Volumes	265
12.1.7	Efficiency boosters: Replication and caching	266
12.2	Security differences between AFS and NFSv4	266
12.2.1	Security and authorization in AFS	266
12.2.2	Security in NFSv4	267
12.2.3	Migration considerations	268
12.3	Migrating AFS users to NFSv4	269
12.4	Migrating AFS groups to NFSv4	270
12.5	Comparing an AFS “cell” and an NFS “domain”	272
12.6	File system semantics	272
12.6.1	AFS implements save on close	272
12.6.2	Difference between AFS and NFS	273
12.7	Building a namespace	273

12.7.1 Pseudo file system	273
12.7.2 External namespace (exname)	273
12.7.3 Referrals and replication	274
12.8 Migrating AFS data to NFSv4 servers	276
12.8.1 Migration options	276
12.8.2 NFS/AFS Translator	278
12.9 Access control lists	278
12.9.1 AFS ACL permissions	279
12.9.2 NFS ACL permissions	280
12.9.3 Detailed comparison of AFS and NFS ACLs	280
12.9.4 Example of an AFS to NFS ACL conversion	282
Chapter 13. Illustrated AFS migration	285
13.1 Introduction	286
13.2 Existing AFS cell setup	286
13.3 Setting the NFS domain to the AFS cell name	288
13.4 Setting up the KRB5/LDAP environment	290
13.5 Migrating users to Kerberos and LDAP	290
13.6 Migrating group information	293
13.7 Migrating data	298
13.8 Migrating ACLs	300
13.9 Accessing the migrated data from NFSv4 clients	305
Part 5. Appendixes	307
Appendix A. Test environment	309
Appendix B. Case study: IBM Global Storage Architecture	313
Business problem	314
Solution	315
GPFS File system	317
Security	318
Load balancing	319
Server hardware	320
Storage	321
Protocols and software	322
Backups	322
Time synchronization	323
Kerberos and NFSv4	323
Centralization	324
Scalability	324
Benefits of GSA File	325
GSA File status	326

Appendix C. Configuring Network Time Service	329
Configuring the NTP server with a reference clock	330
Configuring the NTP server without a reference clock	333
Configuring NTP clients	336
Appendix D. AIX 5L V5.3 NFS quick reference	339
NFS configuration files	340
NFS daemons	340
NFS commands	341
Export options	342
mount command options	343
nfso command options and examples	344
nfs4cl command options and examples	347
Appendix E. Scripts and configuration files	353
Sample LDAP LDIF file for the KDC realm	354
Script to add users to the KDC	354
DFS to AIXC ACL migration example	356
DFS to NFSv4 migration example	361
AFS to Kerberos/LDAP user migration	368
AFS to Kerberos/LDAP group migration	369
AFS to NFSv4 ACL migration	370
Migrate DCE groups to LDAP	376
Migrate DCE groups to LDAP	378
Copy ACL	382
Appendix F. Installing an AIX 5L maintenance level	385
Obtaining the latest fixes	386
On the Web	386
AIX 10/2005 Update CD	386
Installation tips	386
Installation	386
Verifying the installation	387
Appendix G. Sample migration planning worksheet	389
Appendix H. Additional material	391
Locating the Web material	392
Using the Web material	392
Related publications	393
IBM Redbooks	393
Other publications	393
Online resources	394

How to get IBM Redbooks	395
Help from IBM	395
Index	397

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

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

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

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

AFS®

AIX 5L™

AIX®

CICS®

DB2®

DFS™

DYNIX/ptx®

Encina®

@server®

FlashCopy®

HACMP™

ibm.com®

IBM®

Lotus®


Micro-Partitioning™

MQSeries®

OS/2®

POWER5™

pSeries®

Redbooks (logo) ™

Redbooks™

RS/6000®

Storage Tank™

Tivoli®

TotalStorage®

WebSphere®

xSeries®

The following terms are trademarks of other companies:

CacheFS, Solaris, Sun, Sun Enterprise, Sun Enterprise Authentication Mechanism, Sun Microsystems, SunOS, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows NT, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

The most recent maintenance release of IBM® AIX® 5L™ Version 5.3 includes a significant set of new features added to the Network File System version 4 Protocol (NFSv4) implementation. In 2004, the first IBM Redbook devoted to the topic of NFSv4 implementation in AIX 5L was published: *Securing NFS in AIX: An Introduction to NFS V4 in AIX 5L*, SG24-7204.

This IBM Redbook provides additional up-to-date information to help IBM clients understand and take advantage of the new NFSv4 functions provided by AIX 5L Version 5.3 with the 5300-03 Recommended Maintenance Package.

The NFSv4 implementation in AIX 5L has now expanded to provide core features that make it capable of providing a much broader range of distributed file system services than any prior version of NFS. The scope of this book includes methods for implementing NFSv4 in the enterprise and extensive coverage of methods for how it can potentially be used as a migration target for existing Andrew File System (AFS®) and Distributed Computing Environment (DCE)/Distributed File Service (DFS™)-based enterprise file systems.



The team that wrote this book. Front row L-R: Richard Joltes, Trishali Nayar, Bob Oesterlin
Back row L-R: Chris Almond, Aniket Patel, Gene Curylo

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the IBM International Technical Support Organization, Austin Center.

Gene Curylo is an Advisory I/T Specialist and the Team Lead for IBM Shared Filesystems, Austin Center. He has a total of 18 years of IT industry experience, including the past nine with IBM. His areas of expertise include solution implementations using Distributed Computing Environment (DCE), Distributed File Systems (DFSs), Andrew File Systems (AFSs), NetDispatcher, Samba, Apache, SQL, Linux®, and AIX 5L.

Richard Joltes is an Advisory Software Engineer in DCE/DFS Support and SARPC Development at the IBM Pittsburgh Lab. A former instructor and consultant at Harvard University, Richard has 20 years of industry experience, with the last seven at IBM. He has been an active member of DECUS, Usenix, and other professional organizations and has presented papers about electronic publishing technologies and network management. His areas of expertise include DCE/DFS, Kerberos, Encina®, TCP/IP, software development in C and Perl, Ethernet networks, UNIX® system administration, system security, and Linux.

Trishali Nayar is a Staff Software Engineer at the IBM India Software Labs, Pune. She is currently the Technical Team Leader for the AFS L3 Support Team. She graduated from the University of Pune with a Bachelor's degree in Computer Engineering. She has a total of eight years of IT industry experience, including the past five with IBM. Her areas of expertise include Andrew File System (AFS), Distributed File System (DFS), NFS, and software development in C/C++ on AIX 5L, Sun™ Solaris™, and Microsoft® Windows® platforms.

Bob Oesterlin is a Senior Technical Staff Member in IBM Global Services, located in Rochester, Minnesota. Bob is currently assigned to the IBM Global Account, responsible for the architecture of the IBM Global Account worldwide storage and hosting offerings. He has worked at IBM for 27 years and has more than 20 years of experience architecting and deploying distributed systems and applications. His areas of expertise include DCE/DFS, AFS, NFS, TCP/IP, Samba, and AIX 5L. He has presented at multiple AFS and DCE/DFS conferences. Bob graduated from the University of Minnesota with a Bachelor of Science degree in Electrical Engineering. He is also a member of IEEE, Usenix, SAGE, and the OpenAFS board.

Aniket Patel is the Technical Team Leader at the IBM U.K. UNIX Support Centre. He has eight years of experience in UNIX and has worked at IBM for seven years. Aniket graduated from Kingston University, U.K., with a Bachelor of Science (Honours Degree) in Computer Science. His areas of expertise include UNIX Support on AIX 5L, DYNIX/ptx®, and Linux, NFS, TCP/IP, SNA, X.25, DCE, sendmail, IBM MQSeries®, and the Microsoft Windows operating systems. Aniket leads a team of 15 UNIX specialists providing Level 1 and Level 2 UNIX support to customers across the U.K., Ireland, the Middle East, Egypt, Pakistan, and South Africa. Aniket is a coauthor of a previous IBM Redbook, *Securing NFS in AIX: An Introduction to NFS V4 in AIX 5L Version 5.3*, SG24-7204.

Chris Almond is an ITSO Project Leader and IT Architect based in Austin, Texas, where he specialized in managing development projects focused on Linux and AIX 5L systems engineering. He has a total of 15 years of IT industry experience, including the last six with IBM. His experience includes UNIX/Linux systems engineering, network engineering, IBM Lotus® Domino-based content management solutions, and IBM WebSphere® Portal-based solution design.

Acknowledgments

This is the second IBM Redbook from ITSO that focuses on NFSv4. The Redbook team would like to acknowledge and thank the team of authors that wrote the first book in this series, *Securing NFS in AIX: An Introduction to NFS V4 in AIX 5L Version 5.3*, SG24-7204: Lutz Deneffle, Sridhar Murthy, Aniket Patel, and John Trindle. Their book provided a key reference for us to build upon and for validating our own content development efforts.

A complete and detailed IBM Redbook about a topic such as this would not be possible without generous support and guidance from key staff members in the AIX 5L development organization, as well as other IBMers. The Redbook team would like to acknowledge the following people for their excellent contributions in support of this project:

Carl Burnett, IBM AIX Kernel Architecture Team: For his thoughtful insight and overall guidance in helping us develop a content strategy for this book.

Brian L. McCorkle, IBM AIX NFS Development Team: For his continued patience and assistance while we attempted to better understand the implementation of NFS version 4 on AIX 5L. Brian's support enabled us to meet the content objectives for the book.

Margaret Momberger, IBM Research Server and Storage Systems Architect: For extensive draft review and editing feedback.

Duen-wen Hsiao, IBM AIX NFS Development Team: For technical support during the NFS version 4 scenarios testing and implementation testing.

Ufuk Celikkan, IBM AIX Security Development: For technical support during the NFS version 4 scenarios testing.

Drew Walters, IBM AIX Security Development: For technical support during the NFS version 4 scenarios testing.

Yantian (Tom) Lu, IBM AIX Security Development: For technical support during the NFS version 4 scenarios testing.

Stanley Wood, IBM CIO Office: For his generous contribution of the material in Appendix B, “Case study: IBM Global Storage Architecture” on page 313.

Brian Dixon, IBM GPFS Development Team: For technical support during the NFS version 4 scenarios testing and implementation testing.

Todd DeSantis, IBM AFS Support Team: For reviewing the AFS migration planning and sample scenarios.

Phil Hirsch, DCE/DFS Support: For providing DCE example source code and reviewing draft copies during development of the book.

Dan Bauman, DCE/DFS Support Team Lead: For reviewing draft copies and providing feedback throughout the project.

Laura Stentz, IBM SWG/AIM Distributed File Systems Strategy: For draft review feedback.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an e-mail to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. JN9B Building 905
11501 Burnet Road
Austin, Texas 78758-3493



Part 1

Introduction

In Part 1, we present an overview of shared file systems in the enterprise and the evolution of NFS as a key specification for implementing powerful shared file system services.

Introduction

The goal of this IBM Redbook is to provide a technical planning reference for IT organizations considering an implementation of Network File System version 4 Protocol (NFSv4) on IBM AIX 5L, either as part of a new installation or as part of a migration from Andrew File System (AFS) or Distributed File Service (DFS).

This book includes sample migrations that can be used as a road map for existing installations of AFS and DFS.

1.1 Overview of enterprise file systems

Early computing environments consisted of large, centrally managed systems requiring significant environmental controls. Data was accessed using character-cell terminals directly connected to individual machines; no local storage was available to the user except in the form of offline media such as magnetic tape. This resulted in a simple, centrally administered management environment, but also required users to share processor time and other resources. It was also difficult to provide access to geographically remote systems, and such connections might be extremely slow or otherwise unreliable.

This situation changed significantly in the 1980s, as minicomputers, workstations, and personal computer systems began appearing in the marketplace. Because all these machines contained local disks, data began spreading across the enterprise in what was frequently a relatively unconstrained fashion. This created a more complex management environment for the following reasons:

- ▶ A given piece of data might be found on multiple systems. This wastes precious and expensive disk space and also introduces the problem of revision control. Users could no longer be certain which copy was the most recent, because a file might be found on more than one system at a given time. The problem becomes worse as files proliferate across the enterprise, resulting in many copies that might all contain locally introduced changes.
- ▶ Each workstation or PC might contain critical data requiring proper backup and archiving; this introduces requirements for additional user training in such procedures along with hardware such as a local tape drive and proper storage of media to prevent data loss or theft. Outages to individual workstations holding important data might impact overall business activities, and the probability of a user workstation or personal computer outage due to hardware failure or user error is much higher than that of a highly available, managed server system.
- ▶ Lacking a centralized infrastructure available to all client hardware, users might find it difficult to locate resources such as applications, shared hardware, and data files. Therefore, a great deal of time is wasted in attempts to locate a particular file, printer, or other resources.
- ▶ Management of an increasing number of remote systems becomes progressively difficult. Each machine requires hardware maintenance, operating system and other software upgrades, and individual copies of licensed software.

- ▶ Distribution of corporate data might involve shipping tapes or other media to remote offices, or copying files over a network link on a periodic basis. Again, such files invariably will become out of sync with the master copies and might even be outdated before arriving at their destination. This method also introduces the possibility of data loss or theft during the period when the physical media is in transit to its destination.
- ▶ Lacking a central authentication authority, users might require individual login IDs for numerous systems, requiring additional management tasks in the form of account maintenance. This also introduces additional complexity in the area of access management, because a given user might require different levels of authorization to data and other resources located on each system.
- ▶ The presence of many individual per-machine user accounts also might create a fragmented environment in regard to electronic mail and other enterprise-wide applications. For example, unless a central service is installed, it is necessary for users to address mail messages to users at specific machine addresses rather than using a global directory service.

These situations generated a great deal of research into distributed computing technologies. The areas of namespace management, remote access to file systems and other resources, replication and other high availability technologies, centralized backup and data archiving, and authorization received particular attention for the reasons noted here.

A *network* or *distributed* file system is a collection of servers and storage devices that are dispersed across machines on a network. Activity to the storage devices must be carried out across the network. Instead of a single centralized data repository, the file system consists of multiple, independent storage devices. The configuration of a distributed file system can vary. Servers can run on dedicated machines, while other machines can be both a server and a client.

Early versions of these *enterprise file systems* did not operate well in wide area network (WAN) environments. These file systems were designed for use on fast local area networks (LANs), and the long latencies of WANs greatly impacted their performance. The design point for these file systems was the small workgroup, and their security was weak for this reason. Support for replication and location independence was also limited.

1.2 The migration landscape today

As this book was being written, limited choices were available for users of older enterprise file systems such as AFS and DFS. Many of the advanced capabilities of AFS and DFS were not present in previous versions of NFS. This includes replication, caching (referral), and a unified namespace.

For an introduction to AFS and DFS, see 2.2, “Enterprise file system technologies” on page 13.

Enhancements to NFSv4 that are implemented in AIX 5L Version 5.3 Recommended Maintenance Level 03 (RML03) provide current AFS and DFS users with many of the capabilities required to begin a migration from these technologies to a standards-based enterprise file system. Such a migration better positions an organization to take advantage of changing technologies in servers, storage, and networks.

Not all capabilities provided by AFS and DFS are present in NFSv4. An organization considering a migration must review its requirements and dependence on the features provided by both products in order to determine the proper time for migration. The implementation of pilot migration programs will assist in the assessment of the overall impact.

It is important to realize that there will never be a one-for-one replacement for file systems such as AFS or DFS. It is necessary to change other parts of the infrastructure, including applications, management processes, and end-user expectations, in order to conduct a successful migration. These changes do not need to lead to decreased performance or function, but they must be considered as a necessary consequence of a transition to NFSv4.

NFS is not the only choice. If an existing infrastructure is composed primarily of Microsoft Windows clients, other options are available that might be more appropriate, such as IBM AIX FastConnect, Samba, or Microsoft Windows file servers. Each of these alternatives require different approaches and pose unique challenges. We do not address these alternatives in this book.

1.3 Strategic and business context

Although an enterprise file system is an important component of an information technology infrastructure, it is only part of the solution. Customers are faced with the problem of simplifying and optimizing existing infrastructures. This includes servers, networks, clients, management processes, and applications. The overall goal is to reduce cost and complexity while providing a foundation for growth.

The overall performance of an enterprise file system is directly affected by the larger environment in which it operates. Servers, storage, and network all have impacts. There is a wide range of middleware products, application packages, and custom applications that might need modifications in order to effectively exploit an enterprise file system.

This book focuses on enhancements to NFSv4 that enable it to replace AFS and DFS in many customer environments and migration considerations for moving from environments based on versions earlier than NFS version 4. It is safe to assume that any migration to NFSv4 will likely be part of an overall strategic change to an organization's environment.

A well-designed enterprise file system can be the centerpiece of an organization's IT infrastructure. Before embarking on any migration or introduction of NFSv4, an evaluation must be performed of the organizational role of enterprise file systems and the potential impact on existing applications, services, and network.

IBM faced many of these challenges when planning a migration from AFS and DFS in their internal infrastructure. You might find the approach and solution valuable. For details, see Appendix B, "Case study: IBM Global Storage Architecture" on page 313.

1.4 Why NFSv4?

NFS has evolved into a powerful enterprise file system that enables it to take advantage of today's more powerful servers and storage. Earlier enterprise file systems such as AFS and DFS have architectural limitations that limit their ability to process large files and take advantage of the increased memory and multiprocessor support available in modern servers.

AFS and DFS also require custom client code that must be modified to support new releases of operating systems. This dependency limited the acceptance of AFS and DFS, leading to their fall from mainstream use and ultimately to their withdrawal from the market.

Standards-based with multiple vendor support, NFSv4 offers the ability to quickly deploy an enterprise file system without imposing dependencies on custom code. Because the NFS protocol is a standard, it can interoperate with other clients and platforms offering NFS support.

NFS continues to receive wide vendor support and continued enhancement by IBM and others. This means an investment in NFSv4 today can continue to reap rewards in the future.

1.5 The rest of this book

The rest of this book consists of:

- ▶ Part 1, “Introduction” on page 1
 - Chapter 1, “Introduction” on page 3
 - Chapter 2, “Shared file system concepts and history” on page 11
This chapter describes the evolution and high-level characteristics of enterprise file systems.
- ▶ Part 2, “NFSv4 on AIX 5L V5.3” on page 19
 - Chapter 3, “NFSv4 implementation” on page 21
This chapter reviews the NFSv4 implementation on AIX 5L.
 - Chapter 4, “Using NFSv4 with JFS2 or GPFS” on page 53
This chapter provides best practices guidelines for using NFSv4 with JFS2 or GPFS.
 - Chapter 5, “Using NFSv4 features” on page 81
This chapter provides best practices guidelines to use when implementing NFSv4 in your environment.
- ▶ Part 3, “Preparing to use NFSv4” on page 115
 - Chapter 6, “Building an NFSv4 environment” on page 119
This chapter describes the installation and configuration of a typical NFSv4 environment.
 - Chapter 7, “Migration considerations” on page 161
In this chapter, we discuss the general considerations you need to examine when migrating to NFSv4.
 - Chapter 8, “Migration scenarios” on page 189
This chapter describes the migration scenarios that we provide later in this book.
- ▶ Part 4, “Migrating to NFSv4” on page 193
 - Chapter 9, “NFSv3 to NFSv4 migration” on page 195
In this chapter, we cover the NFSv3 to NFSv4 migration.
 - Chapter 10, “Planning a migration from DFS” on page 223
Here, we discuss planning for a migration from DFS to NFSv4.
 - Chapter 11, “Illustrated DFS migration” on page 247
This chapter describes an example migration from DFS to NFSv4.

- Chapter 12, “Planning a migration from AFS” on page 261
In this chapter, we discuss planning for a migration from AFS to NFSv4.
- Chapter 13, “Illustrated AFS migration” on page 285
Here, we provide an example migration from AFS to NFSv4.
- Part 5, “Appendixes” on page 307
 - Appendix A, “Test environment” on page 309
This appendix provides an overview of the test environment that we constructed in support of writing this book.
 - Appendix B, “Case study: IBM Global Storage Architecture” on page 313
This chapter describes the GSA File solution that IBM developed as part of its internal strategy to replace AFS and DFS in its development infrastructure.
 - Appendix C, “Configuring Network Time Service” on page 329
In this appendix, we describe the steps required to configure the Network Time Service on AIX 5L.
 - Appendix D, “AIX 5L V5.3 NFS quick reference” on page 339
In this appendix, we provide a quick reference to NFSv4 commands on AIX.
 - Appendix E, “Scripts and configuration files” on page 353
This appendix provides example code to support the migration examples described elsewhere in this book.
 - Appendix F, “Installing an AIX 5L maintenance level” on page 385
This appendix contains the steps to install an AIX 5L maintenance level.
 - Appendix G, “Sample migration planning worksheet” on page 389
In this appendix, we provide an example planning template for use in preparing for a migration.

Archived

Shared file system concepts and history

The purpose of this chapter is to introduce the concepts of file systems and discuss the characteristics of enterprise file systems. It is useful to understand aspects of the history of distributed file systems before looking at current scenarios and possible migration paths.

We discuss the following topics in this chapter:

- ▶ Characteristics of enterprise file systems
- ▶ Enterprise file system technologies
- ▶ General considerations when using enterprise file systems

2.1 Characteristics of enterprise file systems

This section describes some of the characteristics of modern enterprise file systems. While enterprise file systems vary in the details of their implementation, they do share some common characteristics that make them a powerful addition to today's business environment.

2.1.1 Replication

Replication provides for full copies of file system data on multiple servers. Clients are aware of the replicas and can switch to another server when the currently accessed server becomes unavailable. Clients can analyze available sites and choose which one to access based on network properties or performance. Replication has classically been provided with read-only or read-mostly data. Ideally, the administrative model allows for a high degree of control over replica content and release of updates. Replication provides the following major benefits:

- ▶ Increased availability without single points of failure
- ▶ Controlled consistency through administrator-initiated updates
- ▶ Better performance through affinity (hosting replicas closer to clients)
- ▶ Better performance through distribution of load across multiple servers
- ▶ Reduced network (especially WAN) traffic
- ▶ Continued access to data when there is loss of connectivity to resources across the WAN
- ▶ A convenient means for controlled geographic distribution of data

2.1.2 Migration

Migration enables the relocation of file system data from one server to another. To be effective, it includes server mechanisms that keep data available (online) during migration events with minimal access delays. Additionally, clients are "migration aware." They recognize migration events, follow the data to its new location, and avoid disruptions or unexpected events when accessing applications. Benefits of migration include:

- ▶ Managing load across servers by moving data away from overloaded systems to less loaded systems
- ▶ Server-off load for replacement, retirement, combining, splitting, or maintenance
- ▶ A means to maintain affinity by moving data when its point of access (consumers) moves

2.1.3 Federated namespace

Providing a single federated rendered namespace that is seen from all users greatly enhances collaboration and sharing of file system data. For maximum value, the namespace can exist across many server systems that might be geographically far apart or reside in different administrative domains and organizations. The management model is centralized with minimal administrative interaction with individual server or client systems. Other valuable namespace features include:

- ▶ Significant administrative control over how the namespace is rendered, including which parts of the namespace are visible
- ▶ Methods for self-defining namespace through standards-based centralized authorities such as the Domain Name System (DNS)
- ▶ Redundancy to protect from single points of failure or loss of connectivity to central resources
- ▶ Integration with data management features to allow physical location transparency within a rendered namespace

2.1.4 Caching

Most network file system client implementations do caching of both data and attributes to improve performance and reduce network traffic. This brings the performance of networked file operations closer to those seen by accessing data on storage local to the client.

2.2 Enterprise file system technologies

We describe some previous technologies that provide for some or all of these characteristics in the following sections.

2.2.1 Sun Network File System (NFS)

Developed originally by Sun Microsystems™ in the early 1980s, Network File System v1 (NFSv1) was used internally by Sun to access and move files over the network between servers. NFS enables servers to mount remote file systems from other servers over the network and allowed local access to the files on that remote file system.

In 1985, NFSv2 was released with the SunOS™ (UNIX) operating system. NFS was a useful tool that became very popular, and several variants were produced by various vendors. The University of California created a free version of NFS in the late 1980s, and a standard protocol was produced with RFC 1094 in 1987.

By the early 1990s, efforts were underway to produce an enhanced version of NFSv2: NFSv3 (RFC 1813). NFSv3 was released in 1995. Its focus was to provide enhancements to performance while maintaining backward compatibility with NFSv2. The basic design of NFS did not significantly change from version 2 to version 3, and it retained major design features such as stateless design, security, and recovery.

In 1998, Sun initiated an effort to design NFSv4 (RFC 3530). This design resulted in significant changes for NFS. The new features incorporated into NFSv4 include:

- ▶ File system namespace
- ▶ Access control lists (ACLs)
- ▶ Improved client caching efficiency
- ▶ Stronger security
- ▶ Stateful design
- ▶ Improved ease of use in respect to the Internet

In that same year, Sun relinquished control over NFS development to the Internet Engineering Task Force (IETF), which then assumed responsibility for further development of the standard.

2.2.2 Andrew File System (AFS)

The Andrew File System (AFS) was developed at the Information Technology Center at Carnegie-Mellon University. The Andrew research project (which included AFS) began in the early 1980s and in 1985 was deployed across the Carnegie-Mellon University campus. Between 1985 and 1989, Carnegie-Mellon University provided the file system to several other research facilities. Transarc Corporation, a startup company spun out of Carnegie-Mellon University and partially funded by IBM in 1989, provided a commercially available version of AFS. Transarc was fully integrated into IBM in 1999.

AFS provides many advantages over NFSv2 and NFSv3 in a large enterprise environment. These include:

- ▶ Client-side caching that reduces network traffic.
- ▶ Global namespace that eliminates the need for user logins on multiple servers and multiple mount points. A derivative of MIT Kerberos IV was used to provide authentication and encryption services.
- ▶ Greatly improved security over public networks.
- ▶ Powerful administrator functions that enable data movement and backup without shutting down user access to the data.

- ▶ Replication capabilities using a single read/write copy backed by one or more read-only duplicates. This paradigm also permits users to read data from a duplicate copy, while reserving the read/write master for operations requiring write access.
- ▶ Clients are provided with an automatic fail-over capability, allowing them to detect the loss of a server and connect to another machine with no user intervention required.

2.2.3 Distributed Computing Environment/Distributed File System

The Distributed Computing Environment (DCE) was designed in the late 1980s by the Open Software Foundation (OSF), which then coordinated subsequent development processes in conjunction with IBM, Digital Equipment Corporation, Apollo Computer, Hewlett-Packard, and Transarc Corporation; the latter additionally took responsibility for development of the Distributed File System (DFS) component. The DCE product fundamentally offers a remote procedure call (RPC) mechanism, which provides the basis for the product's centralized namespace management, encryption and authentication subsystem based on an early release of the MIT Kerberos V5 protocol, Distributed Time Service (DTS), and an application threading mechanism. Version 1.0 was released in 1990, and version 1.2.2 of OSF DCE is still available from The Open Group, the successor to the OSF. This version provides the basis for all current commercial releases of the product, although each vendor's versioning scheme differs from that used by The Open Group.

Like AFS, DFS provides centralized file system creation, management, optimization, replication, and backup services, along with local caching to improve performance across the network. Large installations often involve multiple replicas of critical file systems, distributed across geographically disparate sites in order to serve nearby customers while providing improved fail-over and availability characteristics. DFS extends AFS technology, providing file-level access control, improved encryption and security, and other enhancements including:

- ▶ Improvements to ACL management and granularity.
- ▶ Full POSIX file system semantics, including byte-range file locking.
- ▶ POSIX-based threading.
- ▶ Elimination of the AFS 2 GB file size limitation.
- ▶ Scheduled replication, in addition to the AFS release replication.
- ▶ Better performance with a kernel resident file server.
- ▶ Better security (AFS uses Kerberos IV, while DFS adopted Kerberos V5) and directory services.

- ▶ The more robust Episode file system.
- ▶ Log-based file system, which is more reliable and offers better recovery.

DFS is also an OSF/Open Group product, which is licensed to IBM and other vendors that offer it commercially on a number of platforms.

AFS and DFS offer both automated and manual replication strategies, permitting a great deal of flexibility in the scheduling of changed data to read-only servers. This permits individual sites to configure replication schedules as desired in order to streamline the use of network resources.

2.3 General considerations when using enterprise file systems

Here are some general considerations to remember when using any enterprise file system:

- ▶ Enterprise file systems are designed to provide access to data across multiple platforms and workstations. They are not designed to operate as an API for distributed processing systems. Using an enterprise file system to provide interprocess communications between workstations is a bad idea and will not scale very well. Eventually, the file system is unable to cope with the activity generated by multiple machines reading and writing to the same directory and file system slow downs or outages might occur.
- ▶ The designer of applications that will use an enterprise file system must be aware of the activity that their application will cause in the file system and take appropriate steps to ensure that the users of the tools will not inadvertently cause performance problems. Examples of this include using a directory to log usage of the tool, or allowing the tool to use the file system as temporary space during processing.
- ▶ Avoid running commands that interact with the file system repetitively, recursively, or in loops. Such instances have the potential to generate a lot of traffic to the servers; especially when you consider that this application is running on multiple machines simultaneously, the workload on the servers will grow rapidly.
- ▶ Avoid using a single file or directory to log job activity that might run on multiple systems. This can become a potentially expensive operation on the servers. When multiple clients are reading a file that is receiving constant updates, the file server must keep all the clients updated with the changes. Multiplied over many clients, this can place a heavy burden on the servers, consuming the server's resources.

- ▶ Try to process data locally as much as possible. Ideally, use the file system as a repository. Data should be copied locally, processed and only when processing is completed, and copied back into the enterprise file system.
- ▶ Replicate data that is infrequently modified and widely used. Replication allows for activity to be distributed among many machines, versus a single machine. In addition, it uses less of the server resources, preserving them for other operations. Tools and static data files are good candidates for replication. Because the entire path to the data must be replicated, it is best to plan for replication when the data structure is initially set up.

Archived



Part 2

NFSv4 on AIX 5L V5.3

In this part of the book, we introduce you to the new NFSv4 features added to IBM AIX 5L V5.3 in Recommended Maintenance Level 03. We briefly discuss the features introduced in the initial AIX 5L V5.3 release and discuss the new features in detail. We also show how you can use the new features. For detailed descriptions and implementation considerations about the features introduced in the initial release of AIX 5L V5.3, see the IBM Redbook *Securing NFS in AIX: An Introduction to NFS V4 in AIX 5L Version 5.3*, SG24-7204. You can view or download this book from the following location:

<http://www.redbooks.ibm.com/abstracts/sg247204.html>

Archived

NFSv4 implementation

The main purpose of this chapter is to provide an update about the new features added to IBM AIX 5L V5.3 *Recommended Maintenance Level 03 (RML03)*. First, we present a brief review of NFS version 4 (NFSv4) functionality introduced in the initial release of AIX 5L V5.3. This is followed by a detailed description, with examples, of the features added to AIX 5L V5.3 RML03. For a detailed description of the NFSv4 features introduced in AIX 5L V5.3, see Chapter 2 of *Securing NFS in AIX: An Introduction to NFS V4 in AIX 5L Version 5.3*, SG24-7204, available at:

<http://www.redbooks.ibm.com/abstracts/sg247204.html>

We discuss the following topics in this chapter:

- ▶ Implementation of the NFSv4 protocol in AIX 5L V5.3
- ▶ NFSv4 features supported in the initial AIX 5L V5.3 release
- ▶ Features introduced in AIX 5L V5.3 RML03
- ▶ List of NFSv4 features supported in AIX 5L V5.3

3.1 Implementation of the NFSv4 protocol in AIX 5L V5.3

AIX 5L V5.3 was the first version of AIX to introduce support for NFSv4, while continuing existing support for NFSv2 and NFSv3. The default NFS protocol version used in server exports and client mounts under AIX 5L V5.3 is still version 3. This decision was made to permit an easier migration to AIX 5L V5.3 from previous versions, because few sites were prepared to implement the features provided by NFSv4. The **vers** option can be used with mounts and exports to specify NFS version 4.

Note: In AIX 5L V5.3, the default for exports is still NFSv3. The **vers** option must be explicitly used to define an NFSv4 export.

The initial AIX support for NFSv4 placed an emphasis on security, with support for the optional NFSv4 ACL model when using the AIX enhanced journaled file system (JFS2). Support for managing access from foreign NFSv4 domains was also included. NFSv4 uses the RPCSEC_GSS RPC authentication flavor supporting the Kerberos V5 security mechanism with AIX 5L V5.3. RPCSEC_GSS can also be used with the NFSv3 protocol. For a detailed description of the new features and security considerations, refer to the IBM Redbook *Securing NFS in AIX: An Introduction to NFS V4 in AIX 5L Version 5.3*, SG24-7204.

Note: The AIX enhanced journaled file system is a JFS2 file system with the extended attributes version 2 capability enabled in order to use NFSv4 ACLs.

3.2 NFSv4 features supported in the initial AIX 5L V5.3 release

This section provides a brief overview of the NFSv4 implementation present in the initial release of AIX 5L V5.3, followed by new features introduced in AIX 5L V5.3 RML03.

The mandatory features of the NFSv4 protocol as described in RFC 3530 are supported with the following exceptions:

- ▶ The UTF8 requirements are not fully supported.
- ▶ The LIPKEY and SPKM-3 security mechanisms are not supported with RPCSEC_GSS authentication.

Restriction: Exporting individual files is not supported over NFSv4, thus the lack of support for diskless clients and Network Installation Management (NIM), which requires individual files to be exported.

The initial implementation also included support for the following optional features:

- ▶ NFSv4 ACLs are supported by both the client and server.
- ▶ Support is provided to map principals and file ownership attributes from one NFSv4 domain into another.
- ▶ The AIX5L NFSv4 implementation supports two ACL types in underlying file systems—NFSv4 and AIXC.

3.2.1 External namespace (exname) support

External namespace, or **exname**, is not part of the NFSv4 RFC; it is an *AIX 5L implementation-specific* feature. The **exname** option extends the pseudo file system concept. The external name in an `/etc/exports` file must begin with the `nfsroot` name. But an **exname** export does not need to correspond to the server's root. Figure 3-1 on page 24 and the examples that follow show how the **exname** option can be used.

The objective is to render the view in a manner so that the client sees what is represented by the pseudo root part of Example 3-1 on page 24. On the server, the following file systems are to be exported:

- ▶ `/local/home/`
- ▶ `/local/dept/`
- ▶ `/local/trans/`
- ▶ `/usr/codeshare/ThirdPartyProgs/`

It is also necessary to ensure that the server's local file system is not exposed to clients. How can this be achieved?

Important: The pseudo root on the server should be set to `/exports`. When the server renders the pseudo file system view for the client, any directory or file under the `/exports` directory is hidden. So, if directories and files under `/exports` are to be made available to the clients, they should either be moved to another directory and exported, or a different directory should be chosen as the anchor for the pseudo root.

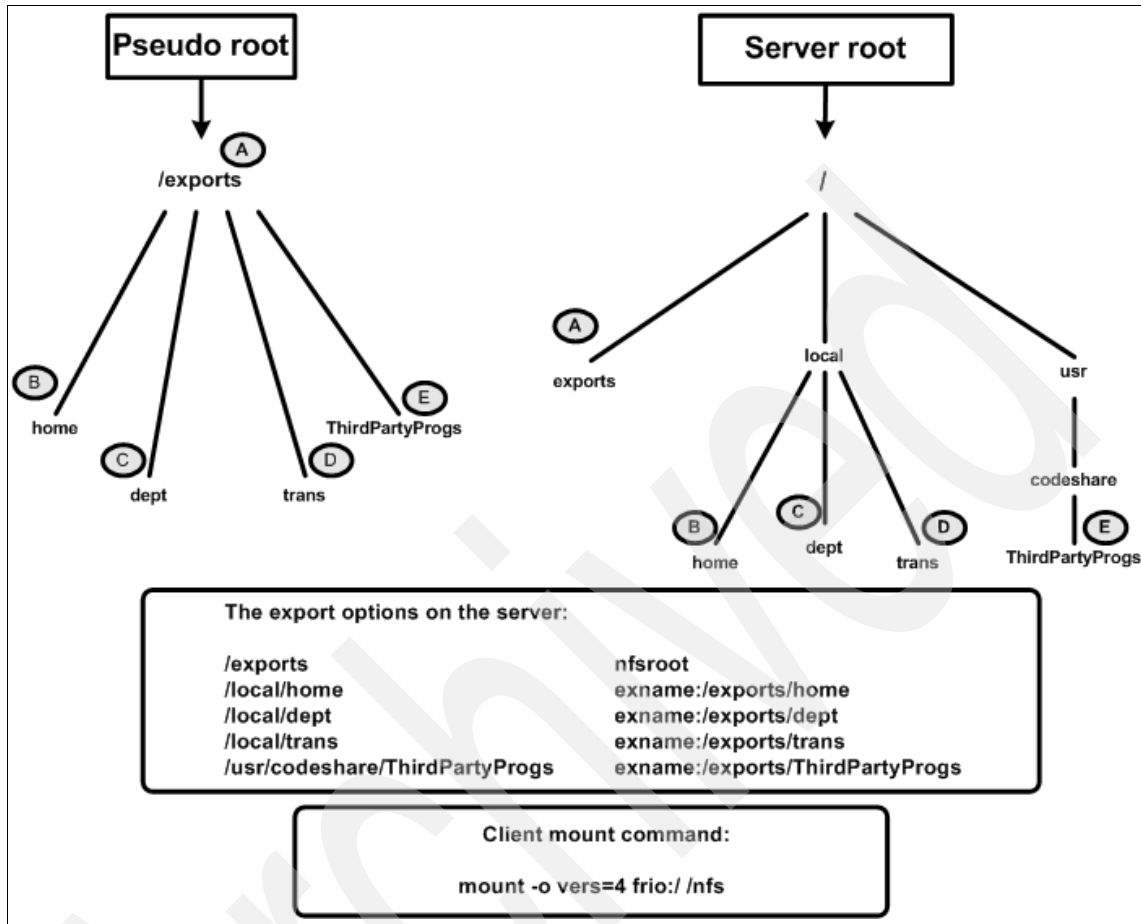


Figure 3-1 Representation of how the server builds the exname pseudo file system view

Using the **exname** option, the `/etc/exports` file is created using the representations shown in Example 3-1.

*Example 3-1 The `/etc/exports` file for rendering the pseudo file system by using **exname***

```
/local/home -vers=4,rw,exname=/exports/home
/local/dept -vers=4,rw,exname=/exports/dept
/local/trans -vers=4,rw,exname=/exports/trans
/usr/codeshare/ThirdPartyProgs -vers=4,ro,exname=/exports/ThirdPartyProgs
```

Example 3-1 shows that the **exname** option does not specify the full path to the individual exports under `/exports`. Although the full path may be provided, by

following this example, the client is never shown the server's directory tree. This hides the actual server file system layout from the client.

Important: We recommend using the **chnfs** command to set the **nfsroot** (nfs root path) and **psuedo-root** (nfs public path). To set the root path, public path, or enable and disable replication and referrals, use following forms of the **chnfs** command:

```
chnfs -r <nfs-root-path>
chnfs -p <nfs-public-path>
chnfs -R <on|off>
```

As another example, suppose a site plans to export a large number of directories under /local, but wants clients to view the exported file systems as part of the /exports tree. The **exname** option can be used as shown in Example 3-2.

Example 3-2 exname exports in the /etc/exports file with full paths

```
/local/trans -vers=4,rw,exname=/exports/local/trans
/local/dept -vers=4,rw,exname=/exports/local/dept
/local/home -vers=4,rw,exname=/exports/local/home
/usr/codeshare/ThirdPartyProgs -vers=4,ro,exname=/exports/local/ThirdPartyProgs
```

3.2.2 FSIDs and file handles

NFSv4 adds the concept of a file system identifier (FSID). The FSID is a 64-bit value that describes an exported container of file system objects (files, directories, and so on) that share the same properties. For UNIX servers, this might typically equate to an exported file system. However, it is server implementation dependent. NFSv4 clients must recognize FSID boundaries and track their properties accordingly. Optional features of the NFSv4 protocol that facilitate replication, migration, and namespace referrals are associated with FSIDs. The AIX 5L V5.3 NFSv4 client recognizes FSIDs and is also capable of tracking and traversing them under a single client-side NFS mount. The **nfs4cl shows** command can be used on the AIX 5L client to examine individual FSIDs and their basic properties.

NFSv4 also introduces volatile file handles, along with the persistent file handles that have existed in previous NFS protocol versions. The server specifies when volatile handles can expire. When they do expire, the client must refresh the file handle using path name information it has recorded. Volatile file handles exist to allow advanced NFSv4 concepts such as replication and migration. They also can make it easier for some platforms to support NFSv4 when the platform's local file system does not make it convenient to maintain persistent handles.

Example 3-3 shows example output from the `nfs4cl showfs` command, run on client system (frio) with four mounts. Notice that the FSID is unique to an individual mount from a specific server, but is not necessarily unique between servers. For example, /mnt2 and /mnt have the same FSID but are from different servers (sabine and trinity, respectively). But /mnt2 cannot have the same FSID as /mnt4, because they are two separate exports from the same server (sabine).

Example 3-3 `nfs4cl` output for NFSv4 client frio

`nfs4cl showfs`

Server	Remote Path	fsid	Local Path
-----	-----	-----	-----
sabine	/gpfs1	2:150	/mnt4
angelina	/gene	10:10	/mnt3
sabine	/	10:4	/mnt2
trinity	/	10:4	/mnt
#			

In the `nfs4cl` command output, the FSID is represented as a *major* and *minor* ID. Therefore, the /mnt4 mount's FSID is major 2 and minor 150.

3.3 Features introduced in AIX 5L V5.3 RML03

AIX 5L V5.3 Recommended Maintenance Level 03 (RML03) further enhances the IBM NFSv4 implementation. This version of AIX 5L adds the following key features to the existing implementation:

- ▶ Delegation
- ▶ Referral
- ▶ Replication

Note: Both servers and clients must be running AIX 5L V5.3 RML03 in order to make use of the new NFSv4 features.

You can download RML03 from the Quick links for AIX fixes Web page:

<http://www.ibm.com/servers/eserver/support/unixservers/aixfixes.html>

Any local IBM Support Center can provide the RML on CD-ROM media.

Execute the command shown in Example 3-4 on page 27 to determine if a system is running AIX 5L V5.3 RML03.

```
# oslevel -r
5300-03
#
```

Example 3-4 shows that the version of AIX installed on this system is AIX 5L V5.3 RML03. If the output from the **oslevel -r** command shows an earlier version of AIX 5L, the required level of AIX 5L or RML03 is not present or has not been correctly installed. Contact your local IBM Support Center to receive assistance in resolving this issue. Appendix F, “Installing an AIX 5L maintenance level” on page 385 describes the procedure for updating a system to RML03 from CD-ROM media or using an image downloaded from the Quick links for AIX fixes Web page.

3.3.1 Delegation

Delegation allows for more efficient cache management by reducing the amount of network traffic associated with cache validation. Client file open and locking activities take place without server calls and are handled by the client using locally cached files. Delegation does not completely eliminate network traffic, because the first *open* call requires contact with the server. The server may grant a delegation to the client for the file. Subsequent opens do not require another call to the server; the delegation is a promise to the client that it can use the file without periodically checking for changes on the server. Instead, the server notifies the client of such activity by rescinding the delegation if another client requests an event that conflicts with the promise, such as modifying the file's data or metadata. The greatest benefit of delegation is for read-only data. The server can grant delegations on read-only files to multiple clients. If the delegation is rescinded, the client can flush its local data and does not need to commit any changes to the server. In AIX 5L V5.3 RML03, delegations are only granted for read-only open requests.

Most NFS client implementations perform caching of both data and attributes to improve performance and reduce network traffic. Some vendors also support a technology known as *cacheFS* that enables extended caching of file and directory content data to persistent storage (disk) on the client system. This further increases the amount of data a client can cache. With caching, some amount of server interaction is still required to conform to required NFS protocol semantics; the client periodically polls the server while files are in use. Depending on the application environment, the network traffic associated with client cache maintenance can be modest. In less reliable or slower networks, this traffic can represent a performance restriction.

NFSv4 provides the optional protocol delegation mechanism, which can improve the caching of NFS. Because the client reads data directly from a local cached copy, the performance is almost as good as working directly from a local file system (because that is the case when the cache is accessed). The reduction in network traffic can help increase the performance and scalability of an NFS environment.

NFSv4, like its predecessors, uses a weak cache consistency model. Clients are not guaranteed to see the most recent changes to data at a server. Delegations are optional and are granted at the NFS server's discretion. Without a delegation, the NFSv4 client operates similar to previous versions of NFS:

- ▶ The client obtains delegation during initial access, for example, open.
- ▶ The client accesses and caches the data.
- ▶ The client can potentially open, lock, read, and close the file with no network traffic.

Delegation is enabled by default for both the client and server, although server delegation is only enabled on the 64-bit AIX kernel. Client-side delegation is supported with both 32-bit and 64-bit kernels. The server only supports read delegation, while the client supports both read and write delegation.

Two new tunable parameters have been added to the **nfso** command. These options allow the control of delegation on the server and client, respectively. The options are:

- ▶ `server_delegation`
- ▶ `client_delegation`

Controlling delegation on the client

Example 3-5 shows how delegation can be controlled on a client. 0 is off, and 1 is on.

Example 3-5 Controlling delegation on the client

```
# nfso -o client_delegation
client_delegation = 1
#
# nfso -o client_delegation=0
Setting client_delegation to 0
#
# nfso -o client_delegation
client_delegation = 0
#
# nfso -o client_delegation=1
Setting client_delegation to 1
#
```



```
# nfsd -o client_delegation
client_delegation = 1
#
```

The **nfs4cl** command can be used to return delegations to the server.

Important: These options will only take effect if set before an NFSv4 volume is mounted. If a file system is already mounted on a client, it is necessary to unmount and remount it before the **nfsd** change can take effect.

Controlling delegation on the server

Delegation on the server can be controlled in one of the two following ways:

- ▶ Using the **nfsd** command.
- ▶ Per export using the new **deleg** option.

Example 3-6 shows how delegation can be controlled using the **nfsd** command. 0 is off, and 1 is on.

Example 3-6 Controlling delegation on the server

```
# nfsd -o server_delegation
server_delegation = 1
#
# nfsd -o server_delegation=0
Setting server_delegation to 0
#
# nfsd -o server_delegation=1
Setting server_delegation to 1
#
```

A more granular level of delegation control can be achieved by using the **deleg** option. This enables delegation to be controlled on a per-export basis and overrides the **nfsd** setting for the export. The **deleg** option is part of the **exportfs** command. The following commands illustrate the use of the **deleg** option within the **/etc/exports** file:

- ▶ Turning delegation on for an export:
`<dir_you_want_to_export> -vers=4,deleg=yes`
- ▶ Turning delegation off for an export:
`<dir_you_want_to_export> -vers=4,deleg=no`

Delegation statistics can be monitored using the **nfsstat -d** command. Example 3-7 on page 30 shows sample output from the command.

Example 3-7 Using the `nfsstat` command to view delegation statistics

```
# nfsstat -d
Server Delegations:
granted      outstanding      recalled      bad recalls
0            0                0            0
returned     long recalls      admin_revoked  resource_recalls
0            0                0            0

Client Delegations:
granted      outstanding      recalled      returned
14           0                4            14
getattns
0
#
```

3.3.2 Referral

The NFSv4 protocol provides functions that enable the distribution of data across multiple servers in a way that is transparent to the users of that data. The first one that we discuss is a global namespace feature called a *referral*. The second feature is a means of specifying locations where copies of data, known as *replicas*, can be found, as discussed 3.3.3, “Replication” on page 36.

A referral is a special NFSv4 object, created in the namespace of a server, to which location information is attached. This server redirects, or refers, operations to the server specified in the location information. In other words, the referral server does not actually contain the file system, but automatically redirects the client to another server that does. Although this might sound simple, it provides a very powerful capability for the location and administration of data.

Figure 3-2 on page 31 shows how referral works. FilesysB does not exist on the primary server. Instead, the primary server’s pseudo root directory contains a reference to the fileset exported on the secondary server. The NFSv4 client is informed of the referral when it receives the pseudo root information from the server, and this enables a user to traverse from a file system on one server to the next file system on the next server seamlessly. As the diagram shows, the user’s view is a single directory structure that appears to be exported from the primary NFSv4 server.

In order to have an NFSv4 server refer to other mounts, the server performing the referral must be enabled for replication. This allows it to maintain location information for the referral. For binary compatibility with previous releases, the default mode is “server does not provide location information,” so referrals must be explicitly enabled. Replication does not need to be enabled on servers that are only referred *to*, that is, those that do not provide referrals. The location

information is a common requirement for both replication (discussed later) and referrals.

Important: In *replication*, the client can have an open state on a replicated file system. This means that it implements NFSv4 features, such as FSIDs, for file descriptors. It will not fail over to a file system possessing different attributes than the one it was using. In this case, enabling replication also (in addition to sharing location information) causes the server to issue volatile file handles. If a fail-over replica server is not replication-enabled, the client will not fail over to a server that is not using volatile file handles. In the *referral* case, the client is never allowed to see any file system information for the referral because there is no file system there, only on the referred-to server. Therefore, there can be no conflict in file system attributes for a referral.

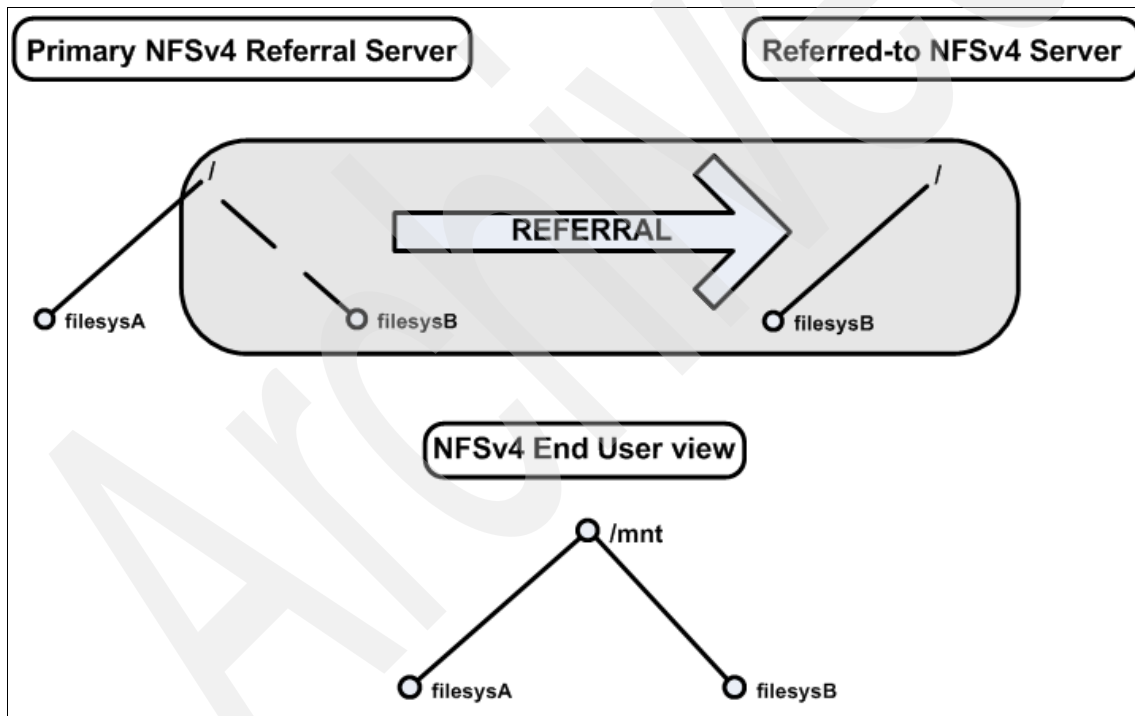


Figure 3-2 NFSv4 referral concepts

An example scenario was created to help better describe the referral process. It begins by first demonstrating how normal NFSv4 exports and mounts work without referrals, and then how they work with them. The syntax for an NFSv4 export in the `/etc/exports` file is as follows:

```
<path_to_dir_to_export> -vers=4,<other_options>
```

The syntax for the NFSv4 mount on the client to mount the *pseudo root* directory from the server is as follows:

```
mount -o vers=4,<other_options> <nfsv4_svr_name>:/ <local_mount_point>
```

Notice that this example mounts the pseudo root directory (`/`) from the server.

Let us see how this maps to our environment:

1. *lavaca* is the main NFS server, and the file system `/work` is to be exported. In order to accomplish this, the following line is added to its `/etc/exports` file:

```
/work -vers=4,rw
```

Tip: The file system `/work` has been exported to all clients in this example. Client access to the directory can be restricted using the **access** option when exporting the file system.

2. The system *lavaca* must now be instructed to export the contents of its `/etc/exports` file:

```
exportfs -va
```

3. On the client, the exported file system is mounted on `/mnt`:

```
mount -o vers=4,rw lavaca:/ /mnt
```

Note: The `/work` export is not being directly mounted from *lavaca* in this step (though there is no prohibition against doing so). The `/work` directory becomes accessible by mounting the pseudo mount (`/`) exported by *lavaca*. This can be seen in Example 3-8. Mounting this pseudo top-level directory is an important step because it provides access to additional exports in the upcoming referral example, without requiring additional steps on the client.

Example 3-8 on page 33 shows the output from the `df -k` command, which displays the mount from *lavaca*. Because the pseudo root directory on *lavaca* was mounted, an `ls` command on `/mnt` shows the exported `/work` directory there.

Example 3-8 *df -k* output showing the NFSv4 mount and *ls* of the mount on the client

```
# df -k
Filesystem
1024-blocks      Free %Used      Iused %Iused Mounted on
/dev/hd4          65536      37500    43%      1651    6% /
/dev/hd2       2195456    1076928    51%     26564    5% /usr
/dev/hd9var       32768      23704    28%       351    5% /var
/dev/hd3          32768      30748     7%        41    1% /tmp
/dev/hd1       311296     301336     4%        21    1% /home
/proc              -          -      -         -      - /proc
/dev/hd10opt      49152      20944    58%       668    6% /opt
lavaca:/          131072     119916     9%       1560    6% /mnt
#
# ls /mnt
work
#
```

When users on the client list the contents of the /mnt directory, they are seeing the /work directory name exported from lavaca.

You can use the **nfs4cl showstat** command on the client to display information about NFSv4 mounts, as shown in Example 3-9.

Example 3-9 *nfs4cl showstat* output on a client system

```
# nfs4cl showstat
Filesystem      512-blocks      Free %Used      Iused %Iused Mounted on
lavaca:/         131072     119916     9%       1560    6% /mnt
#
```

To use a more concrete example, two servers, *lavaca* and *nueces*, have been configured to export NFSv4 volumes. Lavaca is the referring NFSv4 server and is the only server of which end users are aware. Nueces is the referenced server to which lavaca redirects NFSv4 clients requesting access to filesysB.

File systems are laid out in the pseudo root directory on lavaca as follows: filesysA is /work, and filesysB maps to /project.

In this scenario, the user wants to access both the /work directory on lavaca and the /projects directory on nueces. To accomplish this in previous NFS releases, directories must be exported from each server and mounted separately by each client.

The referral feature can be used to simplify this process by:

- Exporting the directories on the referenced servers

- ▶ Creating referrals to those exported directories on the main NFS server
- ▶ Mounting a single pseudo root export from the main server onto the client

As noted earlier, servers that are to issue referrals must have the *replicas* option enabled before the referrals can be created. Here, the referring server is lavaca, where the replicas option has already been enabled. Nueces is only referenced and does not itself reference any other servers. Example 3-10 demonstrates how to enable referrals on lavaca using the NFSv4 replication option.

Example 3-10 Turning on referrals on an NFSv4 server

```
# chnfs -R on
#
# nfsd -getreplicas
replicas=on
#
```

On nueces, the /projects directory is exported by adding the following line to the /etc/exports file:

```
/projects -vers=4,ro
```

Now, the file systems listed in the /etc/exports file are exported using the **exportfs -va** command. This makes /projects available to all NFSv4 clients. Notice that no special preparation is required to export /projects from nueces, because nueces will not be referencing any other servers.

Lavaca can now use referrals to make the /projects export on nueces available to clients. This is accomplished by adding the following line to its existing /etc/exports file:

```
/projects -vers=4,ro,refer=/projects@nueces
```

Example 3-11 shows the full /etc/exports file on lavaca.

Example 3-11 /etc/exports file on lavaca

```
/work -vers=4,rw
/projects -vers=4,ro,refer=/projects@nueces
```

The file systems listed in the /etc/exports file on lavaca are now exported using the same method that was used on nueces (**exportfs -va**).

From the first example, the NFSv4 client had already mounted the /work export from lavaca to /mnt. Therefore, there is no need to unmount and remount. The client's view is dynamically updated to show /projects under the /mnt mount point. The client does not need to perform any mounts to other servers; the

system is automatically made aware that the `/projects` export is being provided by `nueces`. Further referrals to `nueces` or any other NFSv4 server can be added to `lavaca` using the steps shown earlier.

Because a client user is unaware of the actual location of the data, the administrator can redirect clients from one server to another simply by changing the referral statement in the exports file on the server.

Although referrals are created using the **exportfs** command, they are different from data exports. A referral can be created within exported namespaces or in unexported namespaces. In the previous example, the `/projects` referral can be created on `lavaca` even if `/projects` is not exported, although the contents would not be visible to clients if they tried to access the unexported `/projects` tree. This action places the referral within the NFSv4 pseudo namespace.

There is no restriction on the number of referrals that can be created within either the server's NFSv4 pseudo space or within an exported file system. Because a referral does not export any data and only has meaning to the NFSv4 protocol, referrals are available only in NFSv4. Exporting a referral without the **vers=4** option will fail.

Restriction: Although an NFSv4 server can be configured to refer to a server exporting NFSv3 filesets, the resulting referral will not be accessible to clients accessing it through the NFSv4 referral mount. However, if the NFSv3 filesets were re-exported from the server with the **vers=3:4** option, the referral would work properly and the NFSv3 fileset would still be available to NFSv3 clients.

A referral creates a special object at the location specified by the directory parameter. Because access to the object is determined by the client's access to the object's parent directory, most other export options have no meaning and are allowed but ignored. The only exception is the **exname** option, which has the expected behavior.

```
/special/users -vers=4,exname=/exported/users,refer=/users@secrethost
```

The clients that mount `/` from that server see the path `/mnt/users`, which redirects the clients to the `/users` directory on `secrethost` (assuming the referring server's `nfsroot` is `/exported`.) On the referring server, the referral object is created in the local namespace at `/special/users`, so no file or directory can exist there when the export is performed. A special object is created on the server to hold the referral location information. Any directories along the path to the referral will also be created if they do not exist. If the referral is not exported, the referral information will be removed from the object, but the object itself will not be removed. The NFSv4 server will not allow clients to access the resulting stale or orphaned referral object, instead returning an error to clients attempting to do so. The

object can be removed using the `rm` command if desired. A referral can be exported again with new referral information. We do not recommend this because it can take time for clients that previously accessed the referral to realize the location information has changed, though the server touches the referral's parent directory to indicate that information in the directory has been changed. This helps clients realize any information that the client has cached about the directory (and the referral within the directory) has changed and must be refetched, but there is no guarantee how long it will take for the clients to notice.

Tip: Administrators should make sure that the referred-to server will not refer the request back to the same data at the first server, creating a circular referral. In the previous example, a circular referral results if the administrator created a referral on `nueces` at `/projects` that referred to `/projects` on `lavaca`.

3.3.3 Replication

Replication is a means of specifying locations where copies of data can be found. It allows copies of data to be placed on multiple NFSv4 servers and informs NFSv4 clients where the replicas can be located. There are two primary reasons for replicating data:

- ▶ Replicating data improves availability in the face of failure by enabling users and applications to switch from a failed data server to a working one.
- ▶ Replication enables load balancing by having different clients refer to multiple servers.

NFSv4 introduces the concept of *file system IDs* (FSIDs). Each export from an NFSv4 server is given a unique FSID. The FSIDs are volatile in that they change during failover to replica servers. On a failure, a replica server provides a client with its own FSID for the replicated fileset. FSIDs are not file system dependent in that they are provided with any individual export from the NFSv4 server, regardless of whether the exported file system is a separate file system or just a directory in a file system on the server. FSIDs are not provided by a server that only maintains a reference to another server. It is expected that in the future it will be possible to assign an FSID to a subset of a file system and server location information. See 3.2.2, “FSIDs and file handles” on page 25 for additional details about FSIDs.

Fail-over replication

A server can export a file system and specify where a replica of that file system is located. In the event that the primary data server becomes inaccessible to the clients, the clients can use one of the replica servers to continue operations on the replicated file system. The export must be the root of a replicated file system, and the replica locations must also be the roots of file systems. A replica location

is a file system attribute, not a directory or file attribute; replica file systems are assumed to be exact copies of the data on the primary server. The AIX 5L server does not specify how the replica file systems are created from the primary file system or how the data is synchronized. If replicas are specified as read/write, some method of keeping the data on the replicas consistent with the primary file system must be provided. We discuss methods of achieving this consistency in “Synchronizing replicas” on page 44.

Important: In order to replicate a file system in a given environment, all directories mounted above a replicated directory must themselves be replicated. Because of this requirement, care needs to be taken when planning a replication.

Replicas are one or more copies of a file system. Typically, replicas reside on a server or servers separate from the system that provides the read/write master copy. If the primary server becomes unavailable, the client can access the same files from a replica location. Figure 3-3 provides a conceptual view of how replication is achieved. Proper use of replication depends on many factors in each environment, such as the number of clients, number of available servers, level of redundancy required, and load balancing requirements.

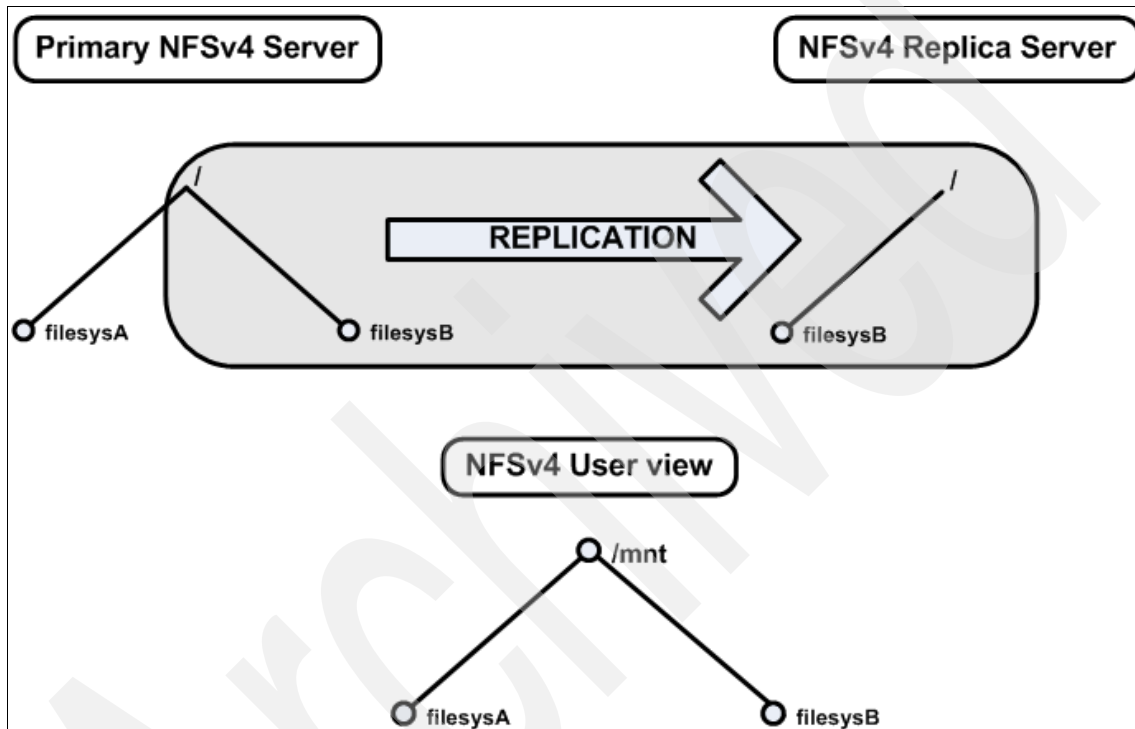


Figure 3-3 NFSv4 replication concepts

Figure 3-4 on page 39 shows a method of configuring an environment to make use of replication. In this case, the loss of SiteA causes clients to fail over to SiteB. If SiteB is lost, clients fail over to SiteC. Additional sites can be added for further resilience, depending on the level of redundancy required. The *sites* shown in the diagram can exist in the same physical location or in distant locations. The concept holds in either case.

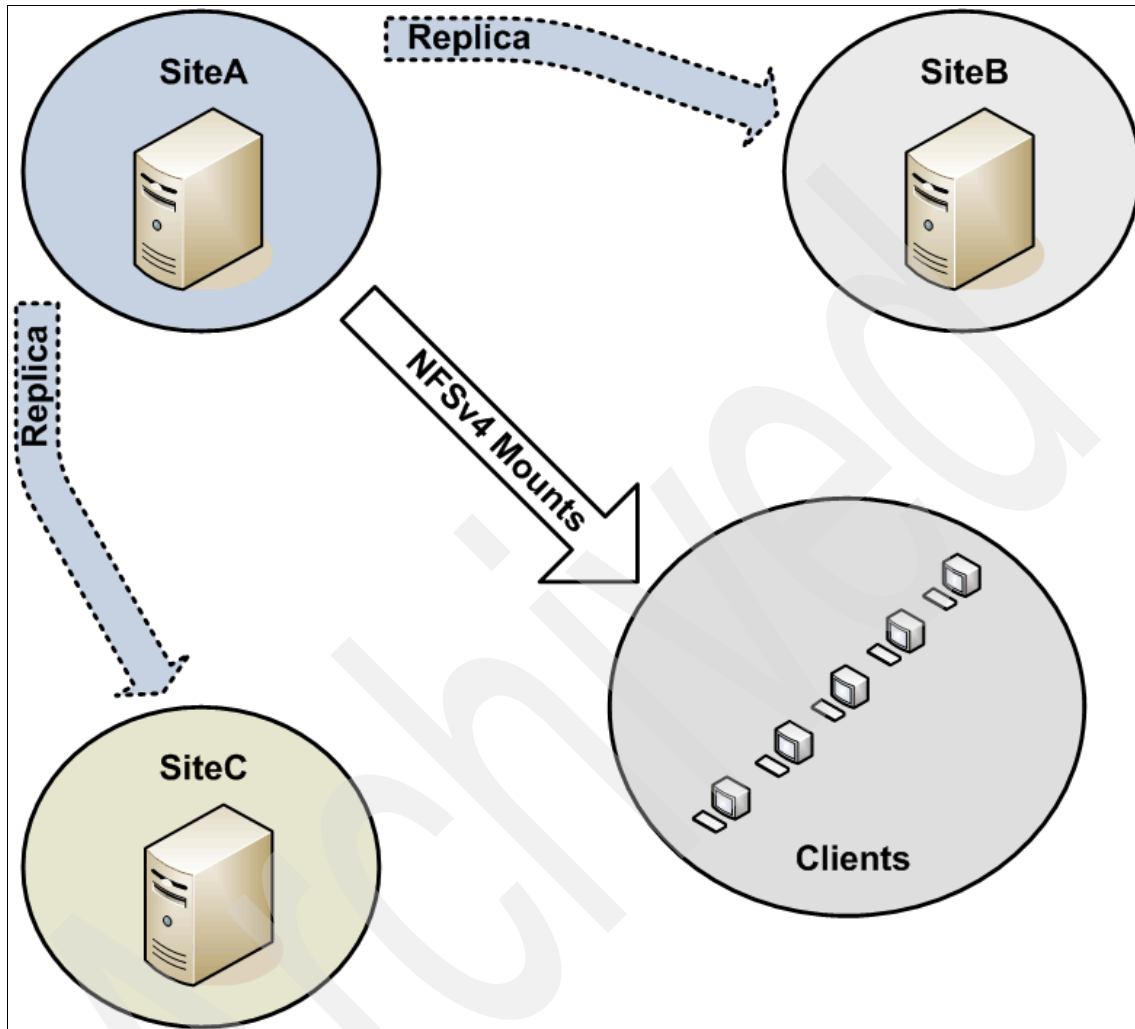


Figure 3-4 Resilient replica environment with fail-over replicas at SiteB and SiteC

Preferred server fallback

After the client failover has occurred, an NFSv4 client running AIX 5L V5.3 RML03 or later will attempt to reconnect (or “fall back”) to the preferred server. The client polling period for fallback to the preferred server is 30 minutes. If the preferred server becomes available again, the client will fall back to that server as soon as it sees it is available again.

Load balancing replication

In a large environment, it does not make sense to point all clients to one NFSv4 server, because this places a substantial strain on that server. The following scenario describes one possible method of balancing load in an NFSv4 environment. See Figure 3-5.

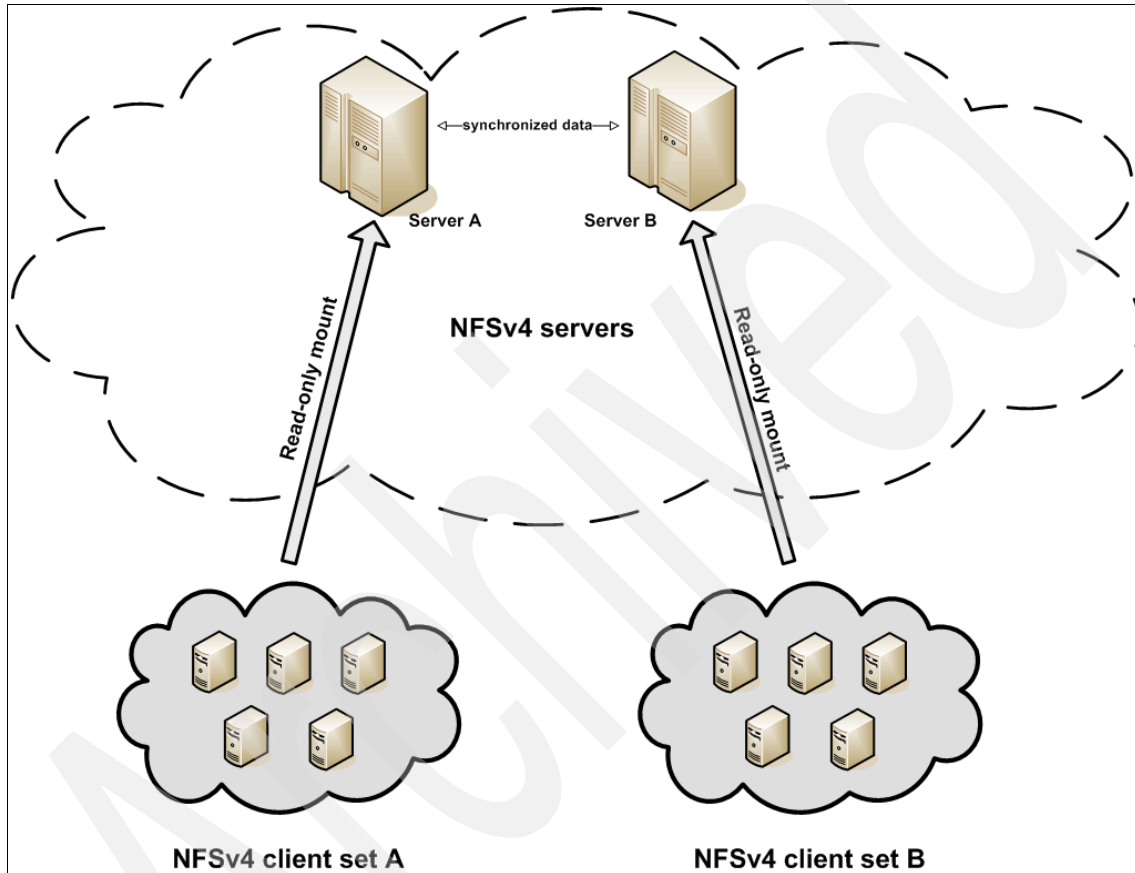


Figure 3-5 Load balancing scenario for replication

Figure 3-5 shows two NFSv4 servers, A and B. Server A has a replica located on server B and has exported file systems with `replicas=/<file system>@ServerB`. Likewise, server B has exported its file system to reflect the fact that server A is a replica server. Clients in client set A mount the file system on server A and fail over to server B if server A becomes unavailable. Clients in client set B mount the file system on server B and fail over to server A if server B becomes unavailable. The data on the two servers are synchronized before they are exported as read-only file systems. The net result is a reduced load on the NFSv4 servers. This scenario can be further expanded by adding additional primary NFSv4

servers and replicas. The replica locations should either explicitly list the local file system as the first replica location, or the administrator should understand that it will be transparently added as the first replica location. In other words, to export on server A:

```
/replicated/database -vers=4,replicas=/replicated/database@serverB
```

This is the same as:

```
/replicated/database \  
-vers=4,replicas=/replicated/database@serverA:/replicated/database@serverB
```

The order of the replicas is important. If the second form is used, but server B is listed first, clients attempting to access /replicated/database on server A are immediately sent to server B.

Before this feature can be used, replication must be enabled on all servers taking part. Example 3-12 shows how replication is enabled on the server.

Example 3-12 Enabling replication

```
# chnfs -R on  
#  
# nfsd -getreplicas  
replicas=on  
#
```

In the previous diagram, if files in the /data directory on server A were also available in the directory /data on server B, NFSv4 clients can be made aware of this by specifying replica locations during the export function. By adding a line similar to the following line to the /etc/exports file for server A, the /data directory is exported while simultaneously specifying the location of another replica:

```
/data -vers=4,ro,replicas=/data@serverA:/data@serverB
```

Tip: Read-only data is ideal for replication. Data can be exported as read/write, but if a failover occurs during transmission, the write might not succeed.

If server A becomes unavailable, users using files under the /data directory of server A will begin using files in the /data directory on server B, without being aware that the client has switched to a different server.

Likewise, the following line shows the entry in the /etc/exports file on server B:

```
/data -vers=4,ro,replicas=/data@serverB:/data@serverA
```

Client-side support for multiple locations

When the client is no longer able to access replicated data on its current server, it then attempts to access the data from the next most favored server. The client creates a preference list when it mounts a file system from the server, using the order specified in the server's `/etc/exports` entry for that file system. This order can be overridden by the client through client-side mount options.

Controlling client replica preferencing

The AIX 5L command `nfs4cl` can be used to control which replica a client will prefer when data is replicated on multiple servers.

Use the `showfs` option available with the `nfs4cl` command to check the client's server preference order, as shown in Example 3-13.

Example 3-13 Displaying current replica preferences

```
# nfs4cl showfs /gpfs1
```

Server	Remote Path	fsid	Local Path
frio	/gpfs1	2:150	/gpfs1
Current Server: frio:/gpfs1			
Replica Server: sabine:/gpfs1			
Replica Server: angelina:/gpfs1			

```
options :  
rw,intr,rsz=32768,wsz=32768,timeo=100,retrans=5,maxgroups=0,acregmin=3,acre  
gmax=60,acdirmin=30,acdirmax=60,minpout=1250,maxpout=2500,sec=sys:krb5:krb5i:kr  
b5p  
#
```

In this case, the client is using the server *frio* as its default, with replicas *sabine* and *angelina*. Use the `nfs4cl setfsoptions` command to change the client preferences, as shown in Example 3-14.

Example 3-14 Changing the preferred client

```
# nfs4cl setfsoptions /gpfs1 prefer=angelina  
#
```

After making the change, the preferred status can again be checked using the `nfs4cl showfs` command, as shown in Example 3-15 on page 43.

Example 3-15 Using the `nfs4cl` command to view fs options

```
# nfs4cl showfs /gpfs1
Server      Remote Path      fsid      Local Path
-----
frio        /gpfs1           2:150     /gpfs1
           Current Server: frio:/gpfs1
           Replica Server: sabine:/gpfs1
           Replica Server: angelina:/gpfs1
options :
rw,intr,rsz=32768,wsz=32768,timeo=100,retrans=5,maxgroups=0,acregmin=3,acregmax=60,acdirmin=30,acdirmax=60,minpout=1250,maxpout=2500,sec=sys:krb5:krb5i:krb5p,prefer=angelina.itsc.austin.ibm.com
#
```

After the client preference has been changed, all subsequent access attempts for data located on a replica file system will be directed to its preferred replica.

Soft mounts and fail-over behavior

By default, NFS establishes *hard* mounts. This means that should a server become unavailable, clients wait indefinitely for the server to come back online. The result for the end user might be a command (such as `ls` against an NFS-mounted directory) that hangs until the server returns to service. NFS supports the *soft* option, which enables a timeout value when a server becomes unresponsive. The result of a timeout in NFSv3 is failed command execution when attempting to access the remote file system. In NFSv4, a failover will result if replicas of the inaccessible file system are available.

The soft mount option in NFS has its own timeout value (`timeo`), and the replica failover option has its own separate timeout value (`nfs_v4_fail_over_timeout`). Furthermore, there is an additional `retrans` mount option for NFSv4 that represents the number of times the client should retry the RPC request to an unresponsive server. The `retrans` option only applies to the `timeo` option for an NFS mount. In other words, the two values are passed to TCP when the client makes an RPC request to the server. The client will initially attempt to contact the server <retrans> number of times for <timeo> tenths of a second. On the first attempt to contact the server, the `nfs_v4_fail_over_timeout` and `timeo` timers are started simultaneously, but the `nfs_v4_fail_over_timeout` is not checked until after the client has completed its <retrans> number of attempts to contact the server for <timeo>. Furthermore, if, after the `retrans` attempts have failed, the `nfs_v4_fail_over_timeout` value has not yet been reached, the access will report a failure and the client will *not* fail over to a replica server. When using soft mounts, take care when setting `nfs_v4_fail_over_timeout` to not use a value larger than `timeo x retrans`.

Synchronizing replicas

Read/write and read-only exports in a replicated environment pose a major challenge in respect to data synchronization. Using Figure 3-4 on page 39 as an example, if SiteA is exporting NFSv4 file systems with the read/write option, then as soon as a client makes an update, the data on SiteB and SiteC will no longer be the same as that on SiteA.

Important: The NFSv4 replication protocol does not provide automatic data synchronization among replica sites.

A method of synchronizing data between sites and replicas is required. Several methods are available for achieving this:

- ▶ Using the **cpio** command, in conjunction with the **rsh** or **ssh** commands
- ▶ Using the **rdist** command
- ▶ Building a clustered General Parallel File System (GPFS) environment

Restriction: The **rdist** command does not preserve NFSv4 ACLs between replicas, so additional work is required to keep ACLs consistent between replica sites.

For small environments, building replicas with the **cpio** command in conjunction with the **rsh** or **ssh** commands is probably the easiest method to implement. This method also preserves the NFSv4 ACLs that might be present on directories or files.

Use of the **rdist** command to build and synchronize replica sites is briefly covered here, but this does not provide a complete solution. A clustered environment using the IBM General Parallel File System (GPFS) provides the most complete solution.

General considerations

You need to make the following decisions before implementing a solution:

- ▶ Which hosts must be kept synchronized?
- ▶ Is there a need to synchronize data, ACLs, or both?
- ▶ Which host is the primary data source?
- ▶ Which directories must be kept synchronized?
- ▶ Should any files or directories be excluded from synchronization?
- ▶ How often should the data be synchronized?
- ▶ What automated method will be used to synchronize the data?

Decisions regarding synchronization methodology can be made after these questions have been answered.

Using the cpio and rsh/ssh commands to synchronize replicas

The AIX 5L **cpio** command provides the ability to back up files and ACLs, and thus can be used in small environments to synchronize data between NFSv4 replicas. In order to preserve extended attributes, the **-U** option must be specified when using the **cpio** command.

In Example 3-16, the systems *brazos* and *trinity* both have JFS2 file systems being exported through NFSv4. The system *trinity* is a read-only replica of *brazos*.

We use the **rsh** and **cpio** commands to synchronize the **/work** directory on *trinity* with the master copy on *brazos*.

Example 3-16 Using the cpio command to copy ACLs and files to a replica

```
# cd /work
#
# rsh brazos "find /work -print | cpio -oacvU " | cpio -icvUud
/work/test2
/work/test2/.sh_history
/work/test2/me
/work/test2/foobar
/work/test2/.profile
/work/test2/1.new
/work/test2/tempo
/work/test2/1
/work/test2/filename
/work/test2/mary
/work/test2/filet
/work/test2/level
/work/test2/level/myfile
/work/test2/another
/work/test2/1.new2
98 blocks
#
```

Important: In order for this command to work properly, the remote host name must be present in the target host's **/.rhosts** file.

After the command completes, the **/work** file system on *trinity* is a copy of the **/work** file system on *brazos*.

Using the tar command to synchronize replicas

Example 3-17 shows an example of using the **tar** command to synchronize replicas.

Example 3-17 Example using the tar command

```
# rsh brazos "tar -cUpf - /work" | tar -xvUpf -
x /work/junk
x /work/junk/genec
x /work/junk/genec/.sh_history, 1100 bytes, 3 media blocks.
x /work/junk/genec/foobar, 0 bytes, 0 media blocks.
x /work/junk/genec/tempo, 0 bytes, 0 media blocks.
x /work/junk/genec/.profile, 52 bytes, 1 media blocks.
x /work/junk/genec/l, 264 bytes, 1 media blocks.
x /work/junk/genec/me, 8 bytes, 1 media blocks.
x /work/junk/genec/mary, 0 bytes, 0 media blocks.
x /work/junk/genec/filename, 0 bytes, 0 media blocks.
x /work/junk/genec/filet, 18 bytes, 1 media blocks.
x /work/junk/genec/another, 0 bytes, 0 media blocks.
x /work/junk/genec/level
x /work/junk/genec/level/myfile, 0 bytes, 0 media blocks.
#
```

Using the rdist command to synchronize replicas

The **rdist** (remote file distribution) command can be used to maintain identical copies of files on multiple hosts. It preserves the owner, group, mode, and modification times, if possible, and can update running programs. The **rdist** command can receive direction from the following sources:

- ▶ The default distribution file, *distfile*, in the user's \$HOME directory
- ▶ A different distribution file, specified by the **-f** flag
- ▶ Command-line arguments that augment or override variable definitions in the distribution file
- ▶ Command-line arguments that serve as a small distribution file

For further details about **rdist** usage, see the command's man pages.

The rdist distribution file (distfile)

The distribution file specifies the files to copy, destination hosts for distribution, and operations to perform when updating files to be distributed with the **rdist** command.

Each entry in the distribution file has one of the following formats:

VariableName = NameList

Defines variables used in other entries of the distribution file (SourceList, DestinationList, or SubcommandList).

[Label:] SourceList -> DestinationList SubcommandList

Directs the **rdist** command to distribute files named in the SourceList variable to hosts named in the DestinationList variable. Distribution file commands perform additional functions.

[Label:] SourceList :: TimeStampFile SubcommandList

Directs the **rdist** command to update files that have changed since a given date. Distribution file subcommands perform additional functions. Each file specified with the SourceList variable is updated if the file is newer than the time stamp file.

Again, for further details, see the **rdist** command's man pages.

The objective of this example is to synchronize data between two systems. The source system is *trinity* and the destination is *brazos*.

The /work directory on trinity will be replicated on brazos. Additionally, the file BAD_FILE is to be excluded from replication. Example 3-18 shows the distfile created on trinity.

Example 3-18 Distfile used to replicate /work on trinity with /work on brazos

```
HOSTS = ( brazos )
FILES = ( /work )
EXLIB = ( BAD_FILE )

${FILES} -> ${HOSTS}
install -R /work;
except /work/${EXLIB};
```

The /.rhosts file on brazos must be updated to contain a reference to trinity before the distfile in Example 3-18 can be used. If this is not done, the **rdist** command fails with permission denied, as shown in Example 3-19.

Example 3-19 rdist failure because trinity has no access to brazos

```
# rdist -f /tmp/distfile
rdist: Updating the host brazos.
rshd: 0826-813 Permission is denied
```

The **rdist** command can now be used with the **-f <full_path_to_distfile>** option to replicate the contents of /work on trinity with /work on brazos. Example 3-20 shows the output from the test environment.

Example 3-20 Running the rdist command using the customized distfile

```
# rdist -f /tmp/distfile
rdist: Updating the host brazos.
rdist: installing: /work/1
rdist: installing: /work/2
rdist: installing: /work/3
rdist: installing: /work/4
rdist: installing: /work/5
rdist: installing: /work/6
rdist: installing: /work/7
rdist: installing: /work/8
rdist: installing: /work/9
rdist: installing: /work/a
rdist: installing: /work/a/1
rdist: installing: /work/a/2
rdist: installing: /work/a/3
rdist: installing: /work/a/4
rdist: installing: /work/a/5
rdist: installing: /work/a/6
rdist: installing: /work/a/7
rdist: installing: /work/a/8
rdist: installing: /work/a/9
rdist: installing: /work/b
rdist: installing: /work/c
rdist: installing: /work/d
rdist: installing: /work/e
```

Note: The BAD_FILE file was excluded from replication because it was specified in the EXLIB variable.

Now, the **rdist** command must be modified to affect only those files that were changed or updated since the initial **rdist** command was run. See Example 3-21, “Updating only changed and new files and directories” on page 48.

Example 3-21 Updating only changed and new files and directories

```
# rdist -b -f /tmp/distfile
rdist: Updating the host brazos.
rdist: updated /work/1
rdist: updated /work/a/1
rdist: installing: /work/b/new_file
rdist: installing: /work/Z
```

As the example shows, only the files /work/1 and /work/a/1 were updated and /work/b/new_file was created. The directory /work/Z was created. All other files were left untouched, because the copies on both master and replica were identical.

If files are found in /work that do not match /work on trinity, they are removed. See Example 3-22.

Example 3-22 Output from rdist

```
# cat sync_data.out
rdist: Updating the host brazos.
rdist: removed /work/passwd
rdist: removed /work/users.out
#
```

The **-b** option is used to perform a binary comparison; any files that have changed are updated. The **-b** option can also be specified during the initial synchronization.

Automating rdist

The best method of automating the **rdist** command involves the use of the **cron** script. Two options are available:

- ▶ Create a script, called by **cron**, that runs the **rdist** command.
- ▶ Specify the **rdist** command in the crontab file.

The first option is a better choice, because it permits modifications to occur to the script without updating the crontab file. For this example, the script **/usr/local/scripts/sync_data.sh** was created. See Example 3-23.

Example 3-23 The rdist script to be called by cron

```
#!/bin/ksh
#
# Script to synchronize data between the NFSv4 master and replicas.
# This script is ONLY to be called by cron(1).
# 15 August 2005.
#
LOGFILE=/usr/local/scripts/sync_data.out
/usr/bin/rdist -b -f /usr/local/scripts/distfile >>$LOGFILE 2>&1
```

Use the **crontab -e** command to edit the root user's crontab file, adding the line shown in Example 3-24 on page 50.

Example 3-24 Entry added to crontab to synchronize data every hour

```
0 * * * * /usr/local/scripts/sync_data.sh 2>/usr/local/scripts/sync_data.errs
```

In this example, the **sync_data.sh** script is called every hour. The decision of how often data should be synchronized depends on the frequency with which files in the master copy are altered or created.

A discussion of replication scheduling reveals the major drawback involved when using the **rdist** command. Use of this method is neither convenient nor efficient if a large amount of highly volatile data is involved. The amount of network traffic will undoubtedly increase and might lead to other performance-related issues. Also, the **rdist** command cannot be used if the files to be synchronized are larger than 2 gigabytes.

For small to medium environments, where data does not change regularly, the **rdist** method is quick and can be implemented with relative ease.

Note: The amount of work performed when testing this method was limited by time and the available test environment. Before implementing this method in a production setting, it is advisable to test its consequences and performance to ascertain the impact on the systems and the network.

3.4 List of NFSv4 features supported in AIX 5L V5.3

Table 3-1 provides the full list of NFSv4 features supported, including the newest features supported by AIX 5L V5.3 RML03.

Table 3-1 NFSv4 features supported up to and including AIX 5L V5.3 RML03

Feature	Description
Attribute classes	Large set of file attributes available in three different classes: Mandatory, Recommended, and Named.
User name to UID mapping	Represent users and groups as strings.
Better namespace handling	Presence of pseudo root, exname, and refer options.
Better security	Use of RPCSEC_GSS using Kerberos.
Client caching and delegation	Improves performance and reduces network traffic.

Feature	Description
Referrals	Enables multiple servers to present a single unified namespace to client systems.
Replication	Provides better availability if a system fails.
Compound RPC	Uses the network resources efficiently by reducing the total number of RPC transactions.
File locking	Provides better detection and recovery of error conditions.
File handles	Volatile file handles provide better interoperability.
Concurrent I/O (CIO)	Application reads and writes that are issued concurrently can run concurrently without reads blocking for the duration of writes, or the reverse.
Direct I/O (DIO)	Enables applications to perform reads and writes directly to the NFS server, bypassing the NFS client caching layer.

Using NFSv4 with JFS2 or GPFS

We discuss the following topics in this chapter:

- ▶ AIX 5L enhanced journaled file system (JFS2)
- ▶ General Parallel File System (GPFS)
- ▶ Backup considerations

4.1 AIX 5L enhanced journaled file system (JFS2)

AIX was the first UNIX operating system with a journaled file system (JFS), a technology now ubiquitous in the industry. In AIX 5L, a new generation file system, enhanced JFS, or JFS2, was introduced to extend scalability beyond the design of the original product. Starting in AIX 5L V5.1.0, the default file system is JFS for the 32-bit kernel and JFS2 for the 64-bit kernel (this setting can be overridden at AIX 5L installation time in order to choose JFS as the root file system).

Figure 4-1 illustrates NFSv4 with JFS2.

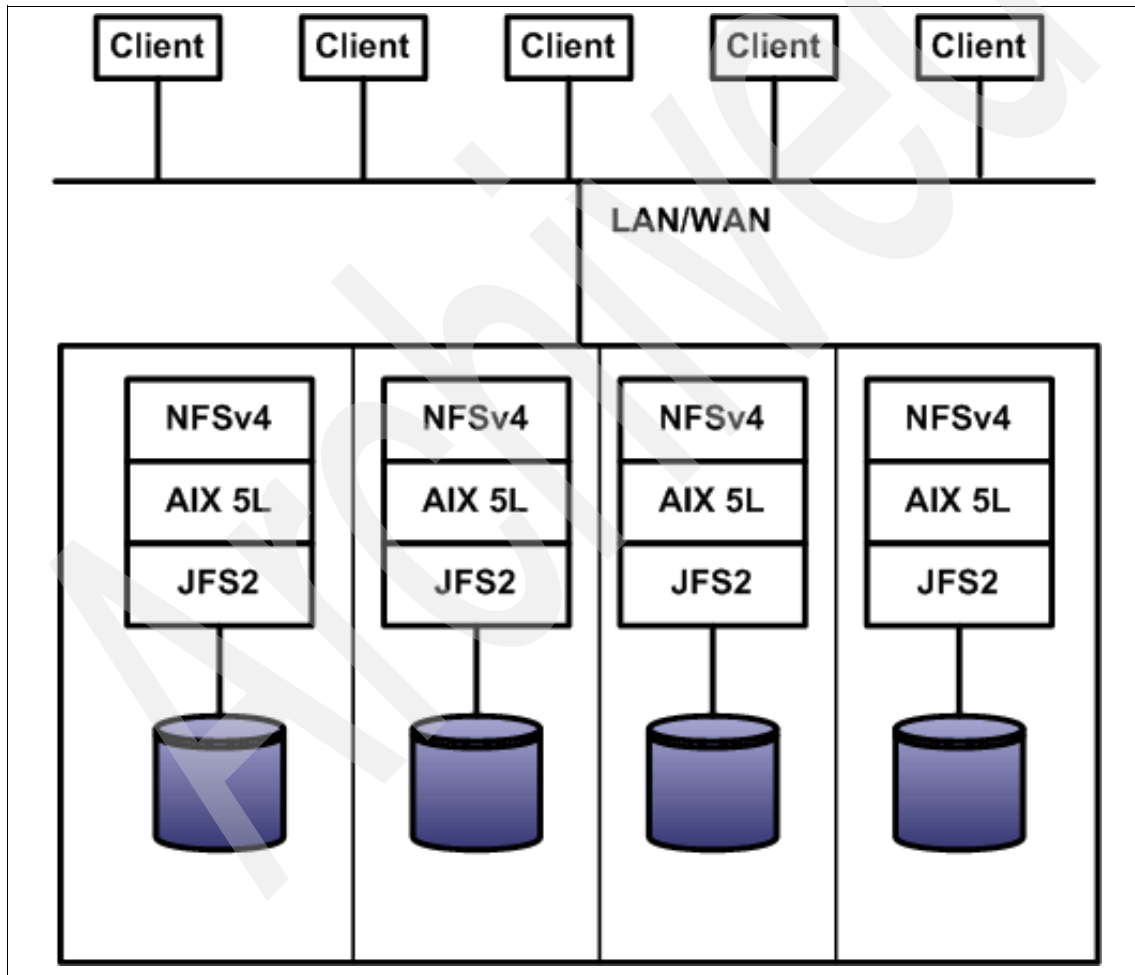


Figure 4-1 NFSv4 with JFS2

JFS2 uses extent-based addressing structures, along with aggressive block allocation policies, to produce compact, efficient, and scalable structures for mapping logical offsets within files to physical addresses on disk. An extent is a sequence of contiguous blocks allocated to a file as a unit and is described by a triple, consisting of logical offset, length, physical address. The addressing structure is a B+ tree populated with extent descriptors (the triples), rooted in the inode, and keyed by logical offset within the file.

JFS2 supports block sizes of 512, 1024, 2048, and 4096 bytes on a per file system basis, enabling users to optimize space utilization based on their application environment. Smaller block sizes reduce the amount of internal fragmentation within files and directories and are more space efficient. However, small blocks can increase path length, because block allocation activities will occur more often than if using a larger block size. The default block size is 4096 bytes, because performance, rather than space utilization, is generally the primary consideration for server systems.

JFS2 dynamically allocates space for disk inodes as required, freeing the space when it is no longer required. This support avoids the traditional approach of reserving a fixed amount of space for disk inodes at file system creation time, thus eliminating the need for customers to estimate the maximum number of files and directories that a file system will contain.

Two different directory organizations are provided. The first organization is used for small directories and stores the directory contents within the directory's inode. This eliminates the need for separate directory block I/O, as well as the need for separate storage allocation. Up to eight entries can be stored inline within the inode, excluding the self (.) and parent (..) directory entries, which are stored in a separate area of the inode.

The second organization is used for larger directories and represents each directory as a B+-tree keyed on name. The intent is to provide faster directory lookup, insertion, and deletion capabilities when compared to traditional unsorted directory organizations.

JFS2 supports the defragmentation of free space in a mounted and actively accessed file system. After a file system's free space has become fragmented, defragmenting the file system allows JFS2 to provide more I/O-efficient disk allocations and to avoid some out of space conditions.

Defragmentation support is provided in two pieces. The first piece is a user space JFS2 utility, which examines the file system's metadata to determine the extent of free space fragmentation and to identify the file system reorganization activities required to reduce or eliminate the fragmentation. The second piece is integrated into the JFS2 kernel extension and is called by the user space utility.

This second piece performs the reorganization activities, under the protection of journaling and with appropriate serialization to maintain file system consistency.

To enhance the performance on a JFS2 file system, a vnode cache has been added and the inode generation numbers have changed.

The problem is that on each access of a file (vnode) by NFS, the vnode and its accompanying inode must be reactivated. Use of a vnode cache keeps these objects in an active state and it becomes much simpler to find and use them. The vnode cache has been adapted from the existing JFS design and implemented in JFS2.

To improve the hash key distribution, the inode generation number has changed. In AIX 5L Version 5.0, the inode generation number started at zero when a file system was mounted, and new inodes got ever-increasing values. In AIX 5L Version 5.1, the inode generation number starts at a number derived from the current time. This results in more non-zero bits and more variation.

JFS2 provides the capability to store much larger files than the existing JFS. JFS2 supports up 4 petabyte (40000 gigabyte) file systems; however, only 16 terabytes have been tested. This applies to file sizes.

JFS2 was designed to give optimal performance with a 64-bit AIX kernel.

File system performance is critical for NFS serving. JFS2 provides major performance gains for NFS. It provides:

- ▶ Very large inode, name, and vnode caches
- ▶ Directory organization
- ▶ Sequential I/O with extents

4.1.1 Comparing JFS2 with JFS

Table 4-1 shows a comparison of JFS2 and JFS.

Table 4-1 Comparison of JFS2 and JFS

Attribute	JFS2	JFS
Architected maximum file system size	4 petabytes (PB)	1 terabyte (TB)
Supported maximum file system size	16 TB	1 TB
Architected maximum file size	4 PB	64 gigabytes (GB)
Supported maximum file size	16 TB	64 GB

Attribute	JFS2	JFS
Max inodes	Dynamic, limited by available blocks	Configured at file system creation time
Available file system block sizes	512 bytes (B), 1 kilobyte (KB), 2 KB, 4 KB	512 B, 1 KB, 2 KB, 4 KB, 128 KB

4.1.2 JFS2 advanced features

JFS2 supports the following advanced features:

- ▶ Direct I/O (DIO)
- ▶ Concurrent I/O (CIO)
- ▶ Snapshots

Direct I/O

Direct I/O (DIO) bypasses file system caches and buffering, allowing for:

- ▶ Direct transfers between disk and application buffer
- ▶ Reduced data copies and CPU processing
- ▶ Performance similar to raw disk I/O

DIO brings benefits to applications that have low data reuse and large sequential I/O.

Concurrent I/O

Concurrent I/O (CIO) alters file system inode level serialization. It is designed for applications with internal serialization such as databases. The ability of CIO to allow concurrent writers and readers to the same file provides for performance increases.

For a detailed discussion, refer to the following white paper:

http://www.ibm.com/servers/aix/whitepapers/db_perf_aix.pdf

JFS2 snapshots

Snapshots are a point-in-time online copy of a file system. Their primary use is for online backups or saving a version of a file system. A snapshot can be taken while the file system is active and in use. Up to 15 snapshots are allowed per file system and they remain persistent across system reboots.

During the creation of a snapshot, the snappedFS (the file system that was used to create the snapshot) will be quiesced and all writes blocked. This ensures that

the snapshot really is a consistent view of the file system at the time of the snapshot. When a snapshot is initially created, only structure information is included. When a write or delete occurs, the affected blocks are copied into the snapshot file system.

Write operations on a snapshot have a performance impact caused by the additional processing required to ensure consistency between the file systems and moving prior versions of updated blocks.

Read operations on the snappedFS remain unaffected, although every read of the snapshot will require a lookup to determine whether the block needed should be read from the snapshot or from the snappedFS. For instance, the block will be read from the snapshot file system if the block has been changed since the snapshot took place. If the block is unchanged since the snapshot, it will be read from the snapped file system. A snapshot, when completed, can be used to make a backup of the file system and is able to guarantee the consistency of the backup image.

This operation makes use of the snapshot map, whose location is stored in the snapshot superblock. The snapshot map logically tracks the state of the blocks in the snapped file system and contains the following details:

- ▶ Block address of blocks that were in use in the snappedFS at the time the snapshot was taken
- ▶ Block address of blocks in the snappedFS that were in use and have subsequently been modified or deleted after the snapshot was created
- ▶ Block address of newly allocated blocks in the snapshot that contain the before image of block that have been deleted or written to

Typically, a snapshot will need 2 to 6% of the space needed for the snappedFS. In the case of a highly active snappedFS, this estimate could rise to 15%, although this is really file system dependent. This space is needed if a block in the snappedFS is either written to or deleted. If this happens, the block is copied to the snapshot. Therefore, in highly active file systems, the space in a snapshot file system can be used quite rapidly. Any blocks associated with new files written after the snapshot was taken will not be copied to the snapshot, because they were not current at the time of the snapshot and therefore not relevant.

If the snapshot runs out of space, the snapshot will be discarded as would any other snapshots associated with the snappedFS. Two possible entries can be created in the AIX error log with either of the following labels: J2_SNAP_FULL or J2_SNAP_EIO. If a snapshot file system fills up before a backup is taken, the backup is not complete and will have to be rerun from a new snapshot, with possibly a larger size, to allow for changes in the snappedFS.

JFS2 file systems from previous versions of AIX 5L are fully supported for snapshot images. Snapshot information is stored in a region of the superblock. It is not possible to mount snapshots on a system running AIX 5L at a version prior to Version 5.2.

4.1.3 Using JFS2 with NFSv4

Starting with AIX 5L V5.3, JFS2 now supports access control lists (ACLs) for NFSv4. This enables you to establish fine-grained access control for file system objects and support inheritance features. In order to take advantage of ACL support for NFSv4, you need to create the JFS2 file system with extended attribute format version 2 (EAv2). If you have file systems that have already been created with extended attribute format version 1 (EAv1), you need to convert to EAv2 first.

The JFS2 supports two ACL types from AIX 5L V5.3:

- ▶ AIXC
- ▶ NFS4

4.1.4 JFS2 ACLs versus NFSv4 ACLs

The AIX Classic (AIXC) ACL type provides for the ACL behavior as defined on previous releases of AIX. This ACL type consists of the regular base mode bits and extended permissions. With extended permissions, you can permit or deny file access to specific individuals or groups without changing the base permissions.

Example 4-1 shows the typical output of the **acledit** command before the converting AIXC ACL to NFS4.

Example 4-1 Output of the acledit command before converting to NFS4

```
# acledit /fs2
*
* ACL_type  AIXC
*
attributes:
base permissions
  owner(root):  rwx
  group(system): r-x
  others:  r-x
extended permissions
  disabled
#
```

Now convert ACL with the **aclconvert** command. Example 4-2 shows the usage of the **aclconvert** command and the output of the **acledit** command after converting ACL to NFS4.

Example 4-2 Converting ACL to NFSv4

```
# aclconvert -t NFS4 /fs2
#
# acledit /fs2
*
* ACL_type   NFS4
**
* Owner: root
* Group: system
*
s:(OWNER@):  d      wpDd
s:(OWNER@):  a      rRWxaAcCo
s:(GROUP@):  a      rx
s:(OWNER@):  d      rwpRWxDdos
s:(GROUP@):  d      rwpRWxDaAdcCos
s:(EVERYONE@): a    rRxac
s:(EVERYONE@): d    wpDd
```

Table 4-2 shows the different access control entry (ACE) types.

Table 4-2 Different ACE types

Key	Description
a	ACE type allows the access described in the access mask.
d	ACE type is denied the access described in the access mask.
l	Audit type ACE.
u	Alarm type ACE.

Table 4-3 shows all possible ACE masks.

Table 4-3 All possible ACE masks

Key	Description
r	Permission to read the data for the file or permission to list the contents for a directory
w	Permission to modify the file's data or permission to add a new file to a directory
p	Permission to append data to a file or permission to create a subdirectory to a directory

Key	Description
R	Permission to read the named attributes of a file
W	Permission to write the named attributes of a file
x	Permission to execute a file
D	Permission to delete a file or directory within a directory
a	The ability to read basic attributes (non-ACLs) of a file (Read Base Attributes)
A	Permission to change basic attributes (non-ACLs) of a file
d	Permission to delete the file
c	Permission to read the ACL
C	Permission to write the ACL
o	Permission to change the owner

AIX Classic ACLs (AIXC) do not support inheritance. This feature is only available with NFSv4 ACLs. When a new file or directory is created and the parent directory has an NFSv4 ACL, an ACL will be established for the new file or directory. If an ACL is established for a child file or directory explicitly, it is decoupled from the parent's ACL. Any changes to the parent's ACL will not affect the child's ACL. In an environment where users are not actively managing ACLs, all the ACLs of a directory's children files will likely have the same ACL. It makes sense to keep one copy of the ACL when the ACL data is common.

The following conditions need to be met for inheritance to be used:

- ▶ It is a JFS2 file system with AIX 5L Version 5.3 or later.
- ▶ Extended attribute version 2 (EA_{v2}) is enabled with the JFS2 file system.
- ▶ NFS4 ACL is applied.

When ACLs are configured with the **acledit** command, fields are provided to define different inheritance flags such as **fi** and **di**. Table 4-4 on page 62 describes the inheritance flags.

Table 4-4 NFSv4 ACL inheritance flags description

Key	Description
fi	Can be placed on a directory and indicates that this ACE should be added to each new <i>non-directory</i> file created.
di	Can be placed on a directory and indicates that this ACE should be added to each <i>new directory</i> file created.
oi	Can be placed on a directory, but does not apply to the directory, only to newly created files and directories as specified by the previous two flags.
ni	For child only, no inheritance for grandchild.

4.1.5 How do we implement inheritance NFSv4 ACLs?

Consider a directory named `/nfs4`. Our objective is to configure its subdirectories and files to inherit its ACLs. Example 4-3 provides a method for achieving this objective.

Example 4-3 Configuring ACLs with inheritance

```
# aclconvert -r NFS4 /nfs4
#
# alcedit /nfs4
* ACL_type  NFS4
*
*
* Owner: user10
* Group: staff
*
s:(OWNER@):  a      rwpRWxDaAdcCs fdi
s:(OWNER@):  d      o fdi
s:(GROUP@):  a      rRxadcS fdi
s:(GROUP@):  d      wpWDACo fdi
s:(EVERYONE@): a      rRxadcS fdi
s:(EVERYONE@): d      wpWDACo
#
# touch /nfs4/file1
#
# aclget /nfs4/file1
*
* ACL_type  NFS4
*
*
* Owner: root
* Group: system
*
```

```
s:(OWNER@): a      rwpRWxDaAdcCs  fidi
s:(OWNER@): d      o          fidi
s:(GROUP@): a      rRxadcS  fidi
s:(GROUP@): d      wpWDACo  fidi
s:(EVERYONE@): a    rRxadcS  fidi
#
```

The /nfs4 directory is first converted to an NFS4 ACL. Next, the ACL can be edited using the **acledit** command. The ACLs are then modified, adding any inheritance flags as required. A new file named file1 is created, and its ACL is verified using the **aclget** command to ensure that its ACL is the same as that on /nfs4.

Note: NFS4 ACL inheritance as implemented in AIX 5L V5.3 is only valid for newly created files and directories under the parent directory. If existing files and subdirectories need to inherit ACLs, use the following command to reset the ACLs on these existing files:

```
aclget <parent_directory> | aclput -R directory
```

4.2 General Parallel File System (GPFS)

IBM General Parallel File System (GPFS) for AIX 5L is a UNIX operating system file system designed for clusters of IBM *@server®* pSeries® machines. It enables applications on multiple nodes to share file data with performance and fault tolerance characteristics that surpass those of traditional distributed file systems. GPFS supports very large file systems and stripes data across multiple storage devices or even multiple storage subsystems for higher performance. GPFS is based on a shared disk model that exploits several methods of moving disk data to the application. These methods either involve some form of storage area network (SAN) or software simulation of a SAN using high-speed networking capabilities. This use of disk sharing techniques, combined with a distributed locking protocol to manage the shared disks, produces fewer bottlenecks than other file system techniques. It achieves this by offering the standard AIX file system interfaces, allowing most applications to execute without modification or recompile in addition to providing enhanced parallel programming capabilities in conjunction with the IBM implementation of the Message Passing Interface-I/O (MPI-IO) industry standard. GPFS can be used successfully for a wide range of applications that require high performance, high degrees of scalability, and fault tolerance.

Most UNIX file systems are designed for a single-server environment. In such an environment, adding additional file servers typically does not improve specific file access performance. GPFS is designed to provide high performance by “striping”

I/O across multiple disks, high availability through logging, replication, and high scalability (by using multiple servers) through the IBM RS/6000® SP Switch and the SP Switch2.

GPFS file systems support multiple terabytes of storage within a single file system. As the hardware technology supporting storage attachment matures, it is a logical extension of the GPFS environment to support disks that have a shared direct attachment. GPFS V1.4 begins that process by introducing support for clusters of IBM @server pSeries and IBM RS/6000 systems running HACMP™ and sharing access to disks through Serial Storage Architecture (SSA) links.

At its core, GPFS is a parallel disk file system. The parallel nature of GPFS guarantees that the entire file system is available to all nodes within a defined scope and the file system's services can be safely applied to the same file system on multiple nodes simultaneously. It also means that multiple records can be safely written to the same file simultaneously by being striped across several disks (thus improving performance). This GPFS parallel feature can improve the performance of both parallel and sequential programs. In other words, you do not have to write parallel I/O code to benefit from the GPFS parallelization. GPFS will automatically parallelize the I/O in your sequential program.

In addition to its parallel features, GPFS supports high availability and fault tolerance. The high availability nature of GPFS means that the file system will remain accessible to nodes even when a node in the file system dies. The fault tolerant nature of GPFS means that file data will not be lost even if some disk in the file system fails. These features are provided through integration with other software and hardware, for example, High Availability Cluster MultiProcessing Enhanced Scalability (HACMP/ES) and RAID.

Figure 4-2 on page 65 shows a simplified view of the GPFS architecture with applications.

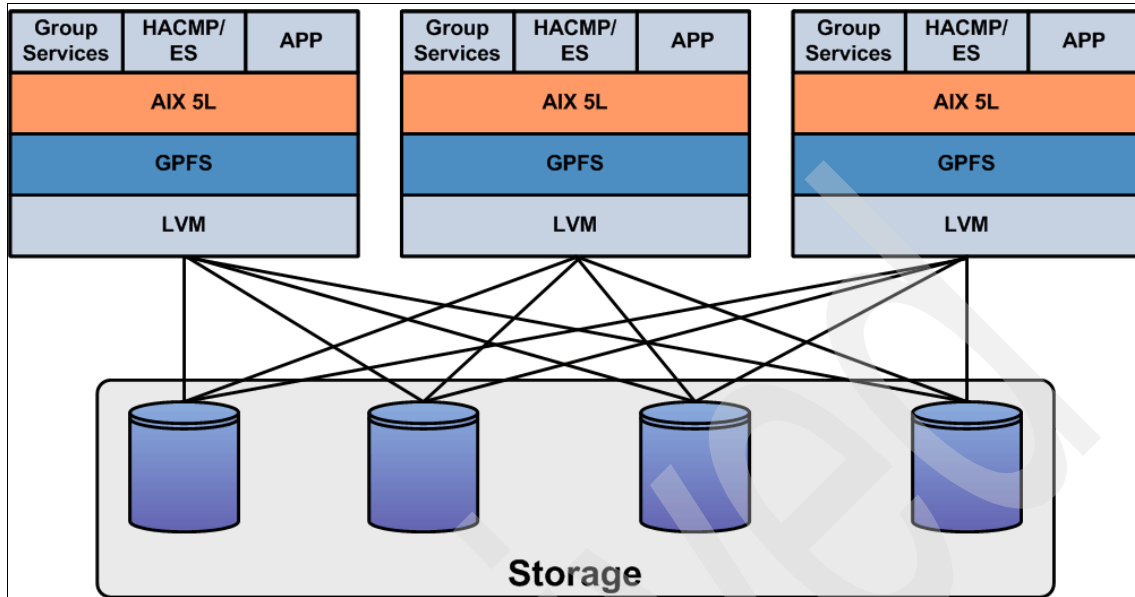


Figure 4-2 GPFS architecture

The file system data and metadata resides on the storage devices assigned to it. Note that this book uses the term *disk* to apply equally to a single disk, a RAID array, or a LUN created by a storage subsystem from pieces of storage that it controls. The installation might choose to allow data and metadata to reside on all disks or to separate metadata to a subset of the disks. In practice, most GPFS installations choose to mix data and metadata, because the density of today's storage media is unfavorable for dedicating large disks to collections of small metadata objects that have potentially high access rates. Mixing the data and metadata allows the entire access capability of all disks in the file system to be brought to bear on a workload that is data intensive at times and metadata intensive at other times. GPFS stripes files across all disks that are defined to the file system, which can result in data blocks from a sufficiently large file being spread across all available disks. Smaller files are striped across the required number of disks; individual disks are selected in an attempt to balance the usage of all disks. This results in high usage of available disk bandwidth to satisfy the needs of applications using very large files and the aggregate needs of numerous applications. GPFS file systems with sufficient storage configurations have delivered multiple GB/sec of bandwidth within a nodeset. The use of multiple disks also allows the creation of very large GPFS file systems. The GPFS product formally supports 100 terabyte (TB) file systems. Larger file systems have been created by special arrangements with IBM.

4.2.1 Why GPFS?

GPFS enables users to group applications based on a business need rather than the location of the data. It is a solution to a requirement for a high-performance cluster file system with fault tolerance characteristics. These characteristics have proved valuable in many types of environments. GPFS has been used in parallel processing environments, wide area file serving, non-database commercial applications, digital media/library applications, and many others.

GPFS is the file system used by many of the world's largest supercomputers. These supercomputers are made up of a large number of powerful multiprocessors linked by a high-performance network. GPFS enables these systems to be used for large parallel jobs that share files across these computers at high data rates. GPFS provides the capability for parallel write sharing of single files with full data consistency. It also provides extremely high aggregate bandwidth for applications or application sets that use independent files concurrently and potentially require the ability to access data assigned to another job instance. It provides these capabilities in an environment that allows redundancy, isolating the application from many failures in the environment. GPFS provides the ability to create large file systems of many terabytes, simplifying the administrative burden of managing multiple file systems.

GPFS provides support for the IBM parallel programming environment and provides extended data access capabilities to the IBM implementation of the Message Passing Interface-I/O (MPI-IO) standard. These functions allow very fine-grained sharing of data among the instances of a parallel application. GPFS allows the MPI-IO facility to give hints that control data prefetching and locking to optimize this parallel use of files. These hints are available for use by other programs with similar requirements.

The same characteristics that make GPFS valuable for high-performance technical applications make it valuable for other applications. Digital media, data mining, engineering, manufacturing applications, and others possess the same requirement for high-speed access to large volumes of data and the ability to access the data from multiple members of a cluster. These applications also benefit from the GPFS fault tolerance abilities, which enable it to keep serving data in the event of a failure. The combination of GPFS and a facility such as HACMP, which redirects application execution to available portion of a cluster when a failure occurs elsewhere, provides excellent reliability.

GPFS clusters provide a capability of doing file serving from a collection of nodes. The file service function can reside on all nodes of the GPFS cluster. This allows the combined resources of multiple nodes in the cluster to be applied to file serving, which results in greater aggregate capability. GPFS will not necessarily produce better single stream response time than single node file systems, but single node file systems are limited in the aggregate to the

capability of the single node. GPFS also provides additional fault tolerance compared to single system solutions.

Figure 4-3 shows one potential implementation of GPFS, a geographically dispersed GPFS environment.

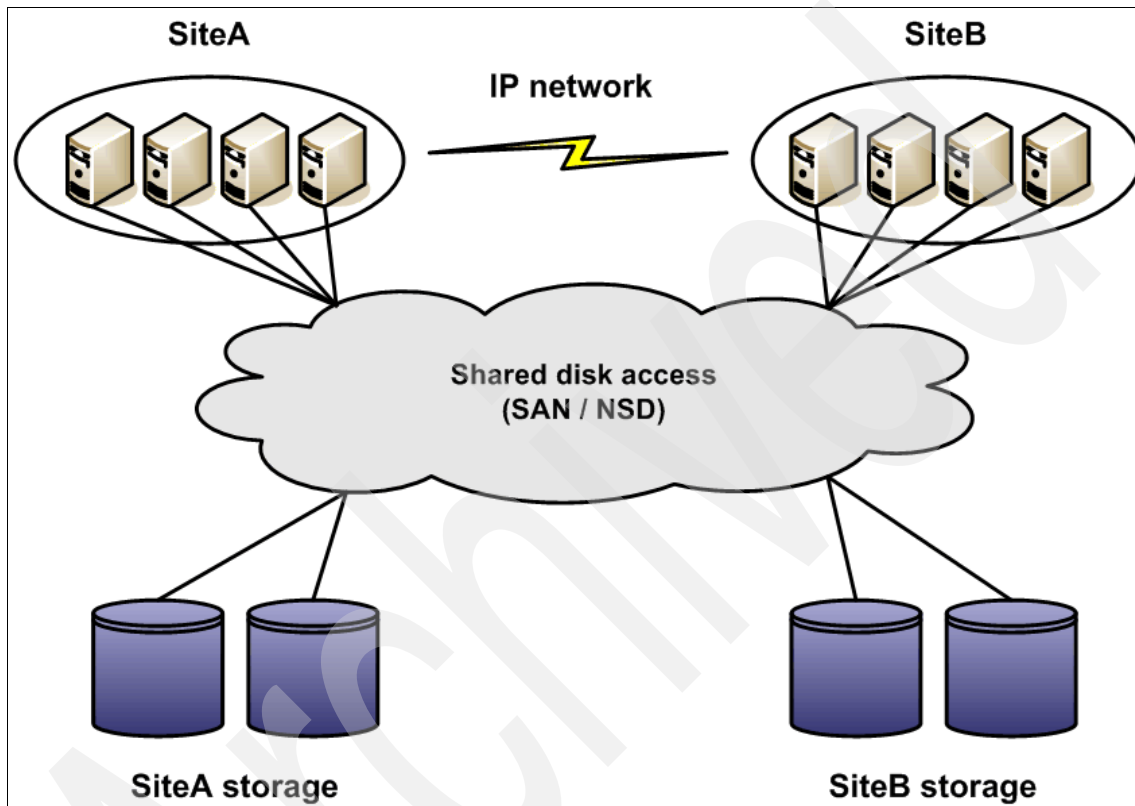


Figure 4-3 A geographically dispersed GPFS environment

4.2.2 GPFS advantages

This section summarizes the key advantages of GPFS, as compared to the file systems considered in the previous section.

Scalability

One of the major advantages offered by GPFS over other file system types is its scalability. GPFS file systems can be striped across multiple disks on multiple storage nodes. This makes it very easy to scale GPFS file systems in terms of both capacity and performance by adding additional nodes as servers, and

additional disk adapters and disks on server nodes. In the GPFS model, each participating system has direct access to the storage containing the striped data. This allows high parallelism and very high I/O bandwidth. This is in contrast to NFS and DFS, which can only serve a file system from the viewpoint of a single server.

Virtual shared disk (VSD) servers do not have to be dedicated but can, in fact, be a GPFS node themselves. This means that client applications can take advantage of spare CPU time on VSD servers. The merits of doing this will, of course, depend on the application and its predicted resource requirements.

Note: Although VSD servers are commonly used in conjunction with GPFS, they are not required in order to run GPFS.

Parallelism

In terms of individual GPFS node to server throughput, overall I/O performance for one node will, of course, not be as great as would I/O to a local JFS. Bigger block size will be better for GPFS performance. However, the first area where GPFS really helps in terms of performance is that the total aggregate I/O throughput for a multiple client application will normally be far in excess of other solutions, such as NFS. This is because the I/O bandwidth can be scaled across multiple servers in order to satisfy the aggregate performance that is required.

The second performance advantage of GPFS is that, because it is a parallel file system, it allows multiple processes to access the same file simultaneously for read, write, or read/write access from different nodes. PIOFS¹ is the only other file system considered in this section that allows parallelism of applications in this way. Applications need to be specifically developed to take advantage of this parallelism, for example, by using the MPI-I/O API.

However, note that while GPFS fully supports advisory byte range locking, PIOFS does not.

Another important performance consideration is that the write I/O of new data blocks is automatically balanced across all disks within the file system. This is due to the GPFS striping algorithm that functions transparently with use of the file system. Traditional local or distributed file systems are far more localized in terms of data placement, which greatly increases the risks of loading imbalances or performance bottlenecks.

Where it is efficient to do so, GPFS automatically invokes read prefetch and write behind algorithms. The algorithms are automatically invoked when GPFS detects

¹ High performance parallel file system. First developed for the SP2 parallel architecture and now superseded by GPFS.

that sequential I/O to a file is being performed. This greatly improves performance for applications that do intensive sequential read and write operations.

Availability

GPFS supports three methods of extended data availability. These methods should be considered according to individual requirements and cost restraints. They are:

- ▶ AIX 5L Logical Volume Manager (LVM) mirroring
- ▶ RAID-1 or RAID-5 disk arrays
- ▶ GPFS replication

The GPFS replication facility permits multiple copies of a file or file system to be maintained automatically. The file system replication factor determines the number of copies it is required to store and can be set for individual files or for the entire file system. Replication is the simplest of the three options to configure but has the greatest performance impact.

Like JFS2, GPFS is also a logging file system. Logs are maintained for each GPFS node, which permit the fast recovery of data in the event of node failure. GPFS uses a token mechanism as an internal mechanism for insuring that data and metadata are consistent and correct.

GPFS supports an automount facility, which means that file systems can be configured to be mounted automatically when the GPFS daemons are successfully started. This increases the overall availability of the GPFS environment.

Simplified administration

GPFS administration is distributed in that a single administration command can be executed on one particular node so that it is also effective on all other GPFS nodes.

With GPFS, a single **mount** command makes it possible to gain access to the entire file system, no matter how many VSD servers it is configured over and whatever its size is. This makes it very easy to move the execution of applications to different nodes. This is in contrast to file systems, where a very large file system might be partitioned across several different server nodes. If there are n GPFS nodes in an application environment, it takes only n mounts to make the GPFS file system available to all nodes, no matter how many VSD servers support the file system.

4.2.3 When to consider GPFS

There are several situations where GPFS is an ideal file system for data on a cluster of pSeries systems:

- ▶ You have large amounts of file data that must be accessed from any node and where you want to more efficiently use your computing resources for either parallel or serial applications.
- ▶ The data rates required for file transfer exceed what can be delivered with other file systems.
- ▶ You require continued access to the data across a number of types of failures.

GPFS is not a wide area distributed file system replacing NFS for network data sharing, although GPFS files can be exported using NFS.

4.2.4 Planning considerations for GPFS

It is not the purpose of this book to discuss GPFS planning in detail. However, we believe the topic needs a mention and you should consider the following areas in detail:

- ▶ Your I/O requirements
- ▶ Planning your hardware layout
- ▶ The GPFS prerequisites
- ▶ The GPFS parameters required to meet your needs

For more information, see the IBM Redpaper *IBM's General Parallel File System (GPFS) 1.4 for AIX*, REDP-0442.

4.2.5 Using NFSv4 with GPFS

Restriction: At the time of writing, it is *not* possible to use the **exportfs** command to NFS export a GPFS file system. NFS development issued APAR IY75298 to address this bug. If you need to use NFSv4 with GPFS, open a call with your local IBM Support Center and request a fix for APAR IY75298. The fix replaces the **exportfs** command and enables you to NFS export a GPFS file system.

Figure 4-4 shows an example of an NFSv4 cluster with GPFS.

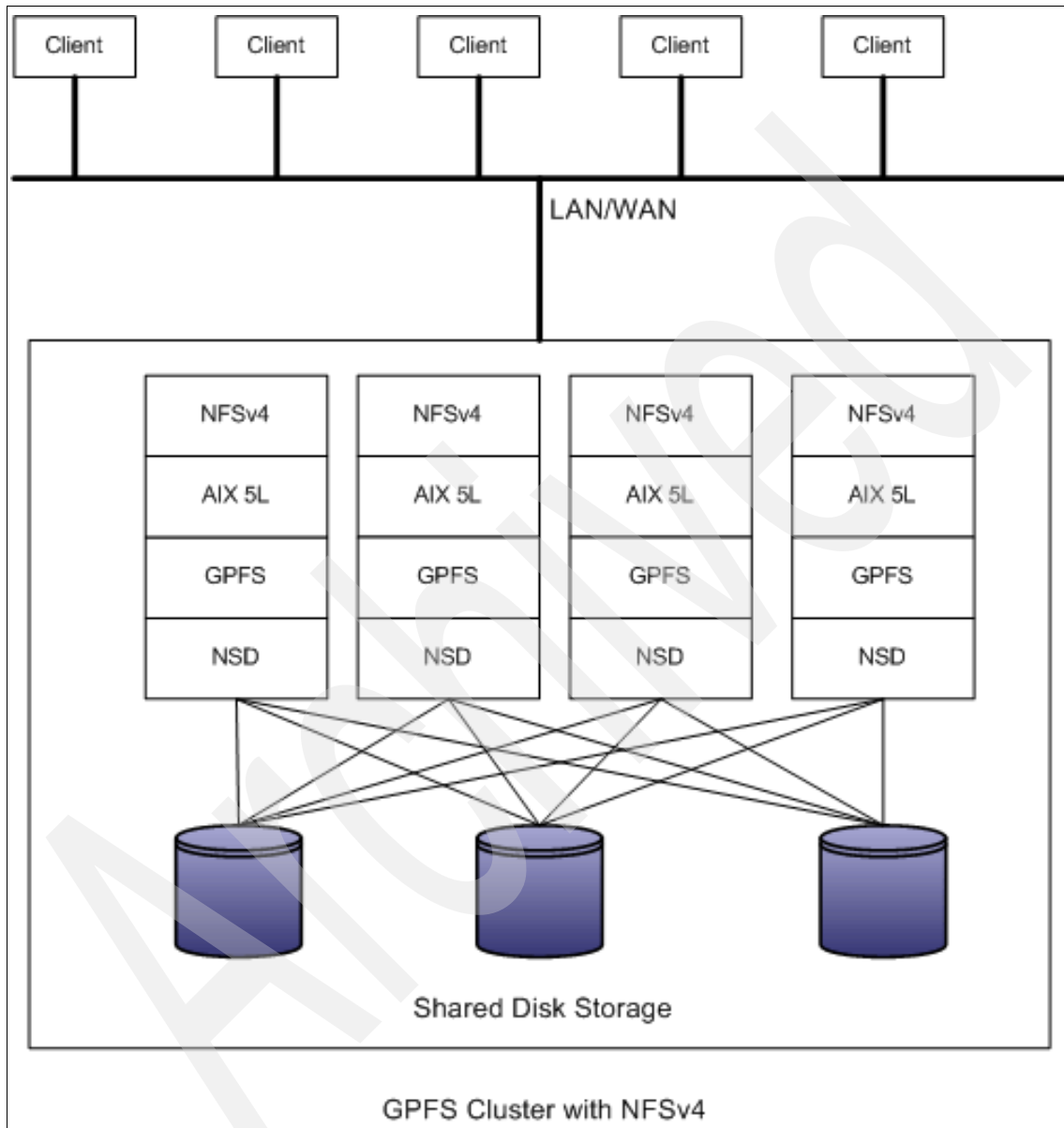


Figure 4-4 An example of an NFSv4 cluster with GPFS

4.2.6 NFSv4 export considerations for GPFS

To export a GPFS file system using NFSv4, there are two file system settings that must be in effect. These attributes can be queried using the `mm1sfs` command and set using the `mmcrfs` and `mmchfs` commands. Consider the following items:

- ▶ The `-D nfs4` flag is required. Conventional NFS access would not be blocked by concurrent file system reads or writes (this is the POSIX semantic). NFSv4 however, not only allows for its requests to block if conflicting activity is happening, it insists on it. Because this is an NFSv4-specific requirement, it has to be set before exporting a file system.
- ▶ The `-k nfs4` or `-k all` flag is required in order to enable NFSv4 ACLs. These flags are not required to NFSv4 export a GPFS file system, but one of them is required if the clients are to have access to the NFSv4 ACLs. See 4.2.8, “NFSv4 ACL administration” on page 72 for more information.

4.2.7 NFS usage of GPFS cache

Exporting a GPFS file system from a node might result in significant additional demands on the resources at that node. Depending on the number of NFS clients, their demands, and specific mount options, you might want to increase either one or both of the `maxFilesToCache` and `pagepool` configuration options. See `mmchconfig` command usage in *General Parallel File System (GPFS) for Clusters: Administration and Programming Reference*, SA22-7967.

4.2.8 NFSv4 ACL administration

AIX 5L does not allow a file system to be NFSv4 exported unless it supports NFSv4 ACLs. The `-k nfs4` or `-k all` flag is required in order to enable NFSv4 ACLs. Initially, a file system will have the `-k posix` setting, and only traditional GPFS ACLs will be allowed (the default for the `mmcrfs` command is `-k posix`).

To export a file system using NFSv4 with support for NFSv4 ACLS, NFSv4 ACLs must be enabled. Because NFSv4 ACLs are vastly different and affect several characteristics of the file system objects (directories and individual files), they must be explicitly enabled. This is done either exclusively by specifying `-k nfs4`, or by allowing all ACL types to be stored (using the `-k all` flag).

After the file system is configured to allow NFSv4 ACLs, it can be exported and NFSv4 clients can mount and use the file system (including setting ACLs that will be in NFSv4 style if you have enabled NFSv4 ACLs).

Depending on the value (`posix | nfs4 | all`) of the `-k` parameter, one or both ACL types can be allowed for a given file system. Because ACLs are assigned on a

per-file basis, this means that within the same file system one file might have an NFSv4 ACL, while another has a POSIX ACL. The type of ACL can be changed by using the `mmputacl` or `mmeditACL` command to assign a new ACL, or by the `mmdeletACL` command (causing the permissions to revert to the mode that is, in effect, a POSIX ACL). At any point in time, only a single ACL can be associated with a file. Access evaluation is done as required by the ACL type associated with the file.

NFSv4 ACLs are represented in a completely different format than traditional ACLs. For detailed information about NFSv4 and its ACLs, refer to *The NFS Version 4 Protocol* paper and other information, available at:

<http://www.nfsv4.org>

In the case of NFSv4 ACLs, there is no concept of a default ACL. Instead, there is a single ACL, and the individual ACL entries can be flagged as being inherited (either by files, directories, both, or neither). Consequently, specifying the `-d` flag on the `mmputacl` command for an NFSv4 ACL is an error.

NFSv4 ACL syntax

An NFSv4 ACL consists of a list of ACL entries. Where traditional ACLs can display one entry per line, the GPFS representation of NFSv4 ACL entries are three lines each, due to the increased number of available permissions beyond the traditional `rwxc`.

The first line has several parts separated by colons (:):

- ▶ The first part identifies the user or group.
- ▶ The second part displays a `rwxc` translation of the permissions that appear on the subsequent two lines.
- ▶ The third part is the ACL type. NFSv4 provides both an allow and deny type:

allow	Allows (or permits) those permissions that have been selected with an X.
--------------	--

deny	Does not allow (or denies) those permissions that have been selected with an X.
-------------	---

- ▶ The fourth and final part is a list of flags indicating inheritance.

Valid flag values are:

FileInherit	Indicates that the ACL entry should be included in the initial ACL for files created in this directory.
--------------------	---

DirInherit	Indicates that the ACL entry should be included in the initial ACL for subdirectories created in this directory (as well as the current directory).
-------------------	---

InheritOnly

Indicates that the current ACL entry should *not* apply to the directory, but *should* be included in the initial ACL for objects created in this directory.

As in traditional ACLs, users and groups are identified by specifying the type and name. For example, group:staff or user:bin. NFSv4 provides for a set of special names that are not associated with a specific local UID or GID. These special names are identified with the keyword special followed by the NFSv4 name. These names are recognized by the fact that they end in with the character @. For example, special:owner@ refers to the owner of the file, special:group@ the owning group, and special:everyone@ applies to all users.

The next two lines provide a list of the available access permissions that can be allowed or denied, based on the ACL type specified on the first line. A permission is selected using an X. Leave the Permissions that are not specified by the entry marked with a minus sign (-).

These are examples of NFSv4 ACLs:

- An ACL entry that explicitly allows READ, EXECUTE, and READ_ATTR to the staff group on a file is similar to the one shown in Example 4-4.

Example 4-4 Example NFSv4 ACL

```
group:staff:r-x::allow
```

```
(X)READ/LIST (-)WRITE/CREATE (-)MKDIR (-)SYNCHRONIZE (-)READ_ACL (X)READ_ATTR (-)READ_NAMED  
(-)DELETE (-)DELETE_CHILD (-)CHOWN (X)EXEC/SEARCH (-)WRITE_ACL (-)WRITE_ATTR (-)WRITE_NAMED
```

- A directory ACL is similar to the one shown in Example 4-5. It can include inherit ACL entries that do not apply to the directory itself, but instead become the initial ACL for any objects created within the directory.

Example 4-5 NFSv4 directory ACL example

```
special:group@:----:deny:DirInherit:InheritOnly
```

```
(X)READ/LIST (-)WRITE/CREATE (-)MKDIR (-)SYNCHRONIZE (-)READ_ACL (X)READ_ATTR (-)READ_NAMED  
(-)DELETE (-)DELETE_CHILD (-)CHOWN (X)EXEC/SEARCH (-)WRITE_ACL (-)WRITE_ATTR (-)WRITE_NAMED
```

- A complete NFSv4 ACL is similar to the one shown in Example 4-6.

Example 4-6 Complete NFSv4 ACL

```
#NFSv4 ACL

#owner:smithj

#group:staff

special:owner@:rwx:allow:FileInherit
(X)READ/LIST (X)WRITE/CREATE (X)MKDIR (-)SYNCHRONIZE (X)READ_ACL (X)READ_ATTR (-)READ_NAMED
(X)DELETE (X)DELETE_CHILD (X)CHOWN (X)EXEC/SEARCH (X)WRITE_ACL (X)WRITE_ATTR (-)WRITE_NAMED

special:owner@:rwx:allow:DirInherit:InheritOnly
(X)READ/LIST (X)WRITE/CREATE (X)MKDIR (-)SYNCHRONIZE (X)READ_ACL (X)READ_ATTR (-)READ_NAMED
(X)DELETE (X)DELETE_CHILD (X)CHOWN (X)EXEC/SEARCH (X)WRITE_ACL (-)WRITE_ATTR (-)WRITE_NAMED

user:smithj:rwx:allow
(X)READ/LIST (X)WRITE/CREATE (X)MKDIR (-)SYNCHRONIZE (X)READ_ACL (X)READ_ATTR (-)READ_NAMED
(X)DELETE (X)DELETE_CHILD (X)CHOWN (X)EXEC/SEARCH (X)WRITE_ACL (-)WRITE_ATTR (-)WRITE_NAMED
```

NFSv4 ACL translation

NFSv4 access requires that an NFSv4 ACL be returned to clients whenever the ACL is read. This means that if a traditional GPFS ACL is associated with the file, a translation to NFSv4 ACL format must be performed when the ACL is read by an NFSv4 client. Because this translation has to be done, an option (**-k nfs4**) is provided on the **mmgetacl** and **mmeditacl** commands so that this translation can be seen locally as well.

It can also be the case that NFSv4 ACLs have been set for some file system objects (directories and individual files) prior to administrator action to revert back to a POSIX-only configuration. Because the NFSv4 access evaluation will no longer be performed, it is desirable for the **mmgetacl** command to return an ACL representative of the evaluation that will now occur (translating NFSv4 ACLs into traditional POSIX style). The **-k posix** option returns the result of this translation.

Users might need to see ACLs in their true form, as well as how they are translated for access evaluations. There are four cases:

- By default, the **mmgetacl** command returns the ACL in a format consistent with the file system setting:
 - If **posix** only, it is shown as a traditional ACL.
 - If **nfs4** only, it is shown as an NFSv4 ACL.
 - If all formats are supported, the ACL is returned in its true form.

- ▶ The command `mmgetacl -k nfs4` always produces an NFSv4 ACL.
- ▶ The command `mmgetacl -k posix` always produces a traditional ACL.
- ▶ The command `mmgetacl -k native` always shows the ACL in its true form, regardless of the file system setting.

In general, we recommend that you continue to use the `mmgetacl` and `mmeditacl` commands without the `-k` flag, allowing the ACL to be presented in a form appropriate for the file system setting. Because the NFSv4 ACLs are more complicated and therefore harder to construct initially, users that want to assign an NFSv4 ACL should use the command `mmeditacl -k nfs4` to start with a translation of the current ACL, and then make any necessary modifications to the NFSv4 ACL that is returned.

Setting NFSv4 access control lists

There is no option on the `mmputacl` command to identify the type (traditional or NFSv4) of ACL that is to be assigned to a file. Instead, the ACL is assumed to be in the traditional format unless the first line of the ACL is:

```
#NFSv4 ACL
```

The lines that follow the first one are then processed according to the rules of the expected ACL type.

An NFSv4 ACL is similar to the one shown in Example 4-7.

Example 4-7 Sample NFSv4 ACL

```
#NFSv4 ACL

#owner:root

#group:system

special:owner@:rwx:allow
(X)READ/LIST (X)WRITE/CREATE (-)MKDIR (X)SYNCHRONIZE (X)READ_ACL (-)READ_ATTR
(-)READ_NAMED
(X)DELETE (-)DELETE_CHILD (-)CHOWN (X)EXEC/SEARCH (X)WRITE_ACL (X)WRITE_ATTR
(-)WRITE_NAMED

special:owner@:----:deny
(-)READ/LIST (-)WRITE/CREATE (-)MKDIR (-)SYNCHRONIZE (-)READ_ACL (-)READ_ATTR
(X)READ_NAMED
(-)DELETE (X)DELETE_CHILD (X)CHOWN (-)EXEC/SEARCH (-)WRITE_ACL (-)WRITE_ATTR
(X)WRITE_NAMED

user:guest:r-xc:allow
```



```
(X)READ/LIST (-)WRITE/CREATE (-)MKDIR (X)SYNCHRONIZE (X)READ_ACL (-)READ_ATTR  
(-)READ_NAMED  
(X)DELETE (-)DELETE_CHILD (-)CHOWN (X)EXEC/SEARCH (X)WRITE_ACL (-)WRITE_ATTR  
(-)WRITE_NAMED
```

```
user:guest:----:deny
```

```
(-)READ/LIST (-)WRITE/CREATE (-)MKDIR (-)SYNCHRONIZE (-)READ_ACL (-)READ_ATTR  
(X)READ_NAMED  
(-)DELETE (X)DELETE_CHILD (X)CHOWN (-)EXEC/SEARCH (-)WRITE_ACL (X)WRITE_ATTR  
(X)WRITE_NAMED
```

This ACL in Example 4-7 on page 76 shows four ACL entries (an allow and deny entry for each of owner@ and guest).

In general, constructing NFSv4 ACLs is more complicated than traditional ACLs. Users new to NFSv4 ACLs might find it useful to start with a traditional ACL and allow either the `mmgetacl` or `mmeditACL` command to provide the NFSv4 translation, using the `-k nfs4` flag as a starting point when creating an ACL for a new file.

Displaying NFSv4 access control lists

The `mmgetacl` command displays an existing ACL regardless of its type (traditional or NFSv4). The format of the ACL that is returned depends on the file system setting (`-k` flag), as well as the format of the actual ACL associated with the file. For details, see “NFSv4 ACL translation” on page 75.

Applying an existing NFSv4 access control lists

This function is identical whether you are using traditional or NFSv4 ACLs.

Changing NFSv4 access control lists

This function is identical whether you are using traditional or NFSv4 ACLs.

Deleting NFSv4 access control lists

Use the `mmdelACL` command to delete NFSv4 ACLs. After the ACL has been deleted, permissions revert to the mode bits. If you then use the `mmgetacl` command to display the ACL (`mmgetacl -k native`), it appears as a traditional GPFS ACL.

When assigning an ACL to a file that already has an NFSv4 ACL, there are some NFS rules that must be followed. Specifically, in the case of a directory, there will not be two separate (access and default) ACLs, as there are with traditional ACLs. NFSv4 requires a single ACL entity and allows individual ACL entries to be flagged if they are to be inherited. Consequently, the `mmputACL -d` command is not allowed if the existing ACL was the NFSv4 type, because this attempts to

change only the default ACL. Likewise the `mmputacl` command (without the `-d` flag) is not allowed because it attempts to change only the access ACL, leaving the default unchanged. To change such an ACL, use the `mmeditacl` command to change the entire ACL as a unit. Alternatively, use the `mmde1acl` command to remove an NFSv4 ACL, followed by the `mmputacl` command.

GPFS exceptions and limitations to NFSv4 ACLs

GPFS has the following exceptions and limitations to the NFSv4 ACLs:

- ▶ Alarm type ACL entries are not supported.
- ▶ Audit type ACL entries are not supported.
- ▶ Inherit entries (`FileInherit` and `DirlInherit`) are always propagated to all child subdirectories. The NFSv4 `ACE4_NO_PROPAGATE_INHERIT_ACE` flag is not supported.
- ▶ Although the NFSv4 ACL specification provides separate controls for `WRITE` and `APPEND`, GPFS will not differentiate between the two. Either both must be specified, or neither can be.
- ▶ Similar to `WRITE` and `APPEND`, NFSv4 allows for separate `ADD_FILE` and `ADD_SUBDIRECTORY` controls. In most cases, GPFS will allow these controls to be specified independently. In the special case where the file system object is a directory and one of its ACL entries specifies both `FileInherit` and `DirlInherit` flags, GPFS cannot support setting `ADD_FILE` without `ADD_SUBDIRECTORY` (or the other way around). When this is intended, we suggest creating separate `FileInherit` and `DirlInherit` entries.
- ▶ Some types of access for which NFSv4 defines controls do not currently exist in GPFS. For these, ACL entries will be accepted and saved, but because there is no corresponding operation, they will have no effect. These include `READ_NAMED`, `WRITE_NAMED`, and `SYNCHRONIZE`.
- ▶ AIX 5L requires that `READ_ACL` and `WRITE_ACL` always be granted to the object owner. Although this contradicts NFSv4 protocol, it is viewed that this is an area where users would otherwise erroneously leave an ACL that only privileged users could change. Because ACLs are themselves file attributes, `READ_ATTR` and `WRITE_ATTR` are similarly granted to the owner. Because it does not make sense to then prevent the owner from accessing the ACL from a non-AIX node, GPFS has implemented this exception everywhere.
- ▶ AIX does not support the use of special name values other than `owner@`, `group@`, and `everyone@`. Therefore, these are the only valid special name for use in GPFS NFSv4 ACLs as well.

4.2.9 NFS client with stale inode data

For performance reasons, some NFS implementations cache file information about the client. Some of the information (for example, file state information such as file size and time stamps) is not kept up-to-date in this cache. The client might view stale inode data (on `ls -l`, for example) if exporting a GPFS file system with NFS. If this is not acceptable for a given installation, caching can be turned off by mounting the file system on the client using the appropriate mount command option.

Note: Turning NFS caching off will result in extra file systems operations to GPFS and negatively affect its performance.

The clocks of all nodes in the GPFS cluster must be synchronized. If this is not done, NFS access to the data, as well as other GPFS file system operations, might be disrupted. NFS relies on metadata time stamps to validate the local operating system cache.

Important: If the same directory is either NFS-exported from more than one node, or is accessed with both the NFS and GPFS mount point, it is critical that clocks on all nodes that access the file system (GPFS nodes and NFS clients) are constantly synchronized using appropriate software (for example, NTP). Failure to do so might result in stale information seen on the NFS clients.

4.3 Backup considerations

Important: As of this writing, IBM Tivoli® Storage Manager Version 5.3 does not support the backup of NFSv4 ACLs on files and directories in JFS2. However, IBM Tivoli Storage Manager Version 5.3 does support the backup of NFSv4 ACLs on files and directories in GPFS.

In order to backup the ACLs associated with your files and directories in JFS2, you must employ additional processes. There are several potential solutions:

- Use the AIX 5L `tar` command.

Using the `tar` command, create one or more backup files for the data and backup the resulting tar file using Tivoli Storage Manager. Make sure that you use the `-U` option on `tar` to back up the extended attributes (ACLs) on the files and directories.

- ▶ Use the AIX 5L **cpio** command.

This is similar to using the **tar** command. Create one or more backup files and then back up resulting file using Tivoli Storage Manager. Make sure that you use the **-U** option with the **cpio** command to back up the extended attributes (ACLs) on the files and directories.

- ▶ Back up data normally using Tivoli Storage Manager and then employ a script to back up the ACLs.

You can continue to use Tivoli Storage Manager to back up your data, but in order to back up the NFSv4 ACLs on files and directories, write a script to capture these ACLs to a file and then back up this file with your data. We do not describe methods for scripting this process in this book.

- ▶ If data resides on a compatible SAN storage device, the FlashCopy® feature can be used to create a data backup.
- ▶ If GPFS is in use, data can be mirrored in order to provide increased reliability.

Using NFSv4 features

We discuss the following topics in this chapter:

- ▶ Using the cache file system (CacheFS)
- ▶ Managing LDAP automount maps
- ▶ Pseudo file system
- ▶ NFSv4 ACLs

5.1 Using the cache file system (CacheFS)

Important: The `bos.net.nfs.cachefs` licensed program product (LPP) must be installed in order to use Sun CacheFS™. This package is located on CD 1 of the AIX 5L V5.3 software. The system must be rebooted following the installation.

The cache file system (CacheFS) is a general-purpose file system caching mechanism that improves NFS server performance and scalability by reducing server and network load. Designed as a layered file system, CacheFS provides the ability to cache one file system on another. In an NFS environment, CacheFS increases the client-per-server ratio, reduces server and network loads, and improves performance for clients on slow links, such as Point-to-Point Protocol (PPP).

Notes:

- ▶ The `/` (root) and `/usr` file systems cannot be cached.
- ▶ Only shared file systems can be mounted using this method.
- ▶ No performance gain is achieved by caching a local journaled file system (JFS) disk.

A cache is created on the client machine, also known as the *front file system*, allowing file systems specified as mounted in the cache to be accessed locally rather than across the network. Files are placed in the cache when a user requests access to them; therefore, initial file requests will not benefit from the cache, while subsequent requests will be faster. Typical NFS speeds will be noted for the initial read request; subsequent reads to data present in the cache occur at performance levels comparable to locally mounted JFS/JFS2 file systems.

5.1.1 CacheFS performance benefits

Note: We abstracted the following information from *AIX 5L Version 5.2 Performance Management Guide*, SC23-4876, available at:

http://www.boulder.ibm.com/pseries/en_US/infocenter/base/aix52.htm

Because NFS data is cached on the local disk after it is read from the server (or *back file system*), read requests to the NFS file system can be satisfied much more quickly than if the data were repeatedly retrieved through the network.

Depending on available memory and the usage pattern of the client, a small amount of data might be retained and retrieved from memory, while the benefit of disk-level caching applies to larger amounts of data that cannot be kept in memory. An additional benefit is that data present in the disk cache is retained at system shutdown, while data cached in volatile memory must be retrieved from the server after a reboot.

Other potential NFS bottlenecks include a slow or busy network and a badly performing or overloaded NFS file server. Therefore, access from the client system to the server is likely to be slow. CacheFS will not prevent the first read from occurring over the network from a particular server, but disk caching helps eliminate or minimize the need for subsequent reads against the same data over the network.

The number of read requests continues to decrease as more cached data become available in the client's local disk cache. This end result is that more clients can be served by a given server, thus the ratio of clients per server is increased. Fewer read requests over the network also permits more effective use of existing bandwidth.

Important: Not every application benefits from CacheFS. Because CacheFS only speeds up read performance, applications that mainly require large or repeated read requests for the same data benefit from CacheFS.

Large CAD applications certainly benefit from CacheFS, because of the often very large models that must be loaded in order to perform calculations.

5.1.2 CacheFS performance impacts

CacheFS does not increase the write performance to NFS file systems. However, you can choose certain write options as parameters to the **-o** option of the **mount** command when mounting a CacheFS. These options influence subsequent read performance to the data. They are as follows:

- ▶ **write around**

The write around mode is the default. It handles writes using the same method used by NFS. Writes are made to the back file system, and the affected file is purged from the cache. This means that write around voids the cache and new data must be retrieved from the server after the write.

► non-shared

The non-shared mode can be used when it is certain no one else will be writing to the cached file system. In this mode, all writes are made to both the front and the back file system, and the file remains in the cache. This means that future read accesses can continue to make use of cached data, rather than accessing the server.

Small reads might be kept in memory, again depending on memory usage, so no benefit is derived from also caching such data on disk. Caching of random reads to different data blocks does not help unless the same data will be accessed repeatedly.

Important: The consistency of the cached data is only checked at specified intervals. Therefore, it is dangerous to cache frequently changed data. Only use CacheFS for read-only or read-mostly data.

5.1.3 Configuring CacheFS

Use the **cfsadmin** command to create the cache, as shown in Example 5-1.

Example 5-1 Creating the cache with cfsadmin

```
# cfsadmin -c /nfscache
# ls /nfscache
.cfs_label      .cfs_lock      .cfs_resource
.cfs_label.dup  .cfs_mnt_points .nsr
#
```

Important: The directory where the cache is to be created must not exist.

This creates a cache file system with the default options. For further information about adjusting the cache size and location, consult the man page for the **cfsadmin** command.

After creating the cache, the cache file system must be mounted and associated with a remote NFS file system. This is done with the **mkcfsmnt** command, as shown in Example 5-2.

Example 5-2 Mounting the CacheFS and associating it with a remote NFS

```
# mkcfsmnt -d /mnt -t nfs -h sabine -p /gpfs1 -c /nfscache -o vers=4,rw
#
# df
Filesystem      512-blocks      Free %Used    Iused %Iused Mounted on
/dev/hd4         131072         101072    23%       1630    13% /
```



```

/dev/hd2      2228224      98216   96%    25053   66% /usr
/dev/hd9var   131072      66560   50%      368    5% /var
/dev/hd3      917504     437392   53%      283    1% /tmp
/dev/hd1      131072     130304    1%        9    1% /home
/proc         -          -        -        -    - /proc
/dev/hd10opt  131072      75448   43%      655    8% /opt
sabine:/gpfs1 2293323776 2292447232 1%      64    1%
/nfscache/.cfs_mnt_points/_gpfs1
sabine:/gpfs1 2293323776 2292447232 1%      64    1% /mnt
#

```

This output indicates that the remote file system has been mounted. It also shows the special mount that was created for the cache file system.

To display information about the cache file system, use the **cfsadmin -l** command, as shown in Example 5-3.

Example 5-3 Using the cfsadmin command to display information about CacheFS

```

# cfsadmin -l /nfscache
cfsadmin: list cache FS information
maxblocks    90%
minblocks    0%
threshblocks  85%
maxfiles     90%
minfiles     0%
threshfiles   85%
maxfilesize  3MB
_gpfs1:_mnt
#

```

For further information about configuring CacheFS, consult the *AIX 5L Version 5.3 Commands Reference*, available at:

<http://publib.boulder.ibm.com/infocenter/pseries/index.jsp?topic=/com.ibm.aix.doc/cmds/aixcmds1/aixcmds1.htm>

5.2 Managing LDAP automount maps

We cover the migration of users from AFS and DFS in other sections of this book. When migration is complete, the administrator must ensure that these users have the ability to log in to any NFSv4 client and gain access to their home directory. A mechanism must be in place to make the users' home directories available on demand. The automount subsystem can be configured to provide this functionality.

If an LDAP environment is already in place, the automount subsystem can be configured to retrieve its maps from an LDAP server by adding the following line into the `/etc/irs.conf` file:

```
automount nis_ldap
```

In order to administer automount maps in LDAP, the appropriate LDIF files must be created. Local automount map files can be converted to LDIF format using the **nistoldif** command. For example, if the LDAP server is named *ldapservers* with its base suffix being *dc=suffix*, the `/etc/auto_home` map file would look like the output shown in Example 5-4.

Example 5-4 The `/etc/auto_home` map file

```
user1 server1:/home/user1
user2 server1:/home/user2
user3 server1:/home/user3
.
.
.
```

Use the following commands to create the LDIF file for `/etc/auto_home` and add it to the LDAP server:

```
nistoldif -d dc=suffix -sa -f /etc/auto_home > /tmp/auto_home.ldif
ldapadd -D cn=admin -w passwd -h ldapservers -f /tmp/auto_home.ldif
```

In order to edit or remove existing automount entries from an LDAP server, the LDIF files must be created manually. For example, create the LDIF shown in Example 5-5 if user2's home directory is on server2.

Example 5-5 LDIF file to change user information

```
# cat /tmp/ch_user2.ldif
dn: automountKey=user2,automountMapName=auto_home,dc=suffix
changetype: modify
replace: automountInformation
automountInformation: server2:/home/user2
#
```

After creating the LDIF shown in Example 5-5, run the following command:

```
ldapmodify -D cn=admin -w passwd -h ldapservers -f /tmp/ch_user2.ldif
```

An LDIF file to remove a user can also be created. For example, create the LDIF shown in Example 5-6 on page 87 in order to remove user3.

Example 5-6 LDIF file to remove user information

```
# cat /tmp/rm_user3.ldif
dn: automountKey=user3,automountMapName=auto_home,dc=suffix
changetype: delete
```

After creating the previous LDIF, you can run the following **ldapmodify** command to delete user3's automount information from LDAP:

```
ldapmodify -D cn=admin -w passwd -h ldapserver -f /tmp/rm_user3.ldif
```

5.3 Pseudo file system

NFSv2 and NFSv3 servers export a set of independent parts of their overall namespace and do not permit NFS clients to cross mount points on the server. This is because NFS expects all lookups to be restricted to a single file system. In NFSv4, the server provides a single root file handle through which clients can obtain file handles for any accessible export.

NFSv4 no longer has a separate *per export* mount protocol. Instead of mounting a number of distinct exports, an NFSv4 client, *using a single mount*, can access the NFSv4 server's exports within a single file tree, called the NFSv4 pseudo file system. The pseudo file system tree, constructed by the server, provides a single, logical view of all the different exported file systems, as shown in Figure 5-1 on page 88.

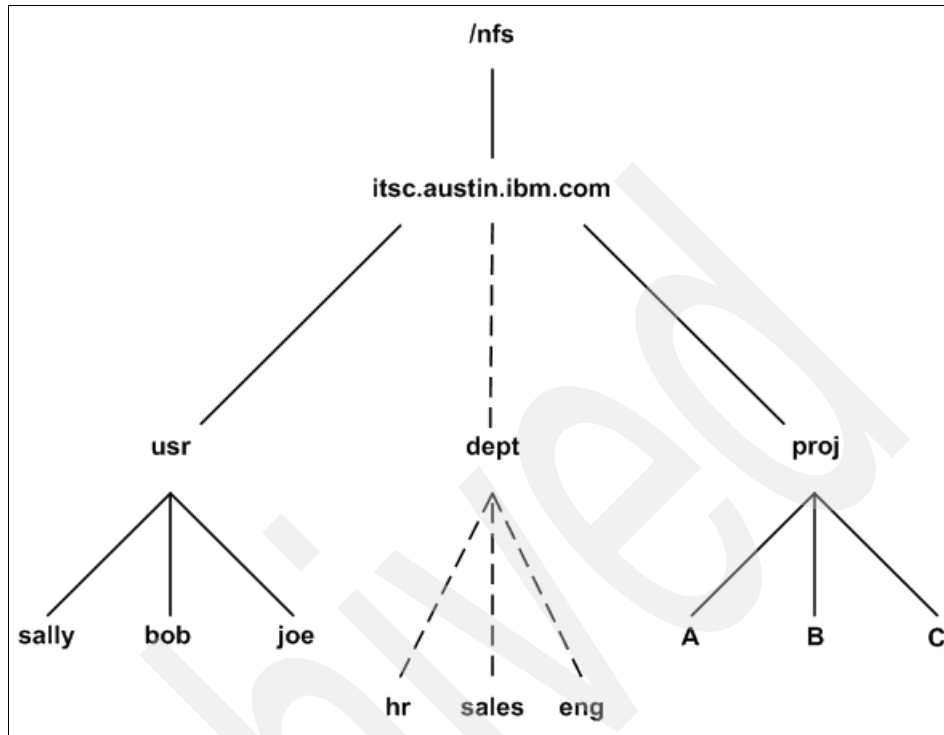


Figure 5-1 Pseudo file system: Server view

For example, a requirement exists to export the following directories:

- ▶ /nfs/itsc.austin.ibm.com/usr/sally
- ▶ /nfs/itsc.austin.ibm.com/usr/bob
- ▶ /nfs/itsc.austin.ibm.com/usr/joe
- ▶ /nfs/itsc.austin.ibm.com/proj/A
- ▶ /nfs/itsc.austin.ibm.com/proj/B
- ▶ /nfs/itsc.austin.ibm.com/proj/C

This requires the following steps on the server:

1. Set the pseudo root node. In this case, it is set to /nfs.
2. Add the directories to be exported to the /etc/exports file, as shown in Example 5-7 on page 89.

Example 5-7 Exporting file systems on the NFS server

```
/nfs/itsc.austin.ibm.com/usr/sally -vers=4,ro
/nfs/itsc.austin.ibm.com/usr/bob -vers=4,ro
/nfs/itsc.austin.ibm.com/usr/joe -vers=4,ro
/nfs/itsc.austin.ibm.com/proj/A -vers=4,ro
/nfs/itsc.austin.ibm.com/proj/B -vers=4,ro
/nfs/itsc.austin.ibm.com/proj/C -vers=4,ro
```

3. Next, execute:

```
exportfs -va
```

4. Mount the root export on the client:

```
mount -o vers=4 <nfsv4_svr_name>:/ /<local_mount_point>
```

Example 5-8 shows a view of the newly mounted file system from the NFSv4 client. Figure 5-2 on page 90 shows a client view of the pseudo file system.

Example 5-8 Client view of the pseudo file system

```
# ls -al /nfs/*
/nfs/itsc.austin.ibm.com/usr:
total 26
drwxr-xr-x  4 root    system      5 Jul 28 11:26 .
drwxr-xr-x  3 root    system      4 Jul 28 11:26 ..
drwxr-xr-x  2 root    system     512 Jul 28 11:24 bob
drwxr-xr-x  2 root    system     512 Jul 28 11:25 joe
drwxr-xr-x  2 root    system     512 Jul 28 11:24 sally
/nfs/itsc.austin.ibm.com/proj:
total 18
drwxr-xr-x  3 root    system      4 Jul 28 11:26 .
drwxr-xr-x  3 root    system      4 Jul 28 11:26 ..
drwxr-xr-x  2 root    system     512 Jul 28 10:40 A
drwxr-xr-x  2 root    system     512 Jul 28 10:40 B
drwxr-xr-x  2 root    system     512 Jul 28 10:40 C
```

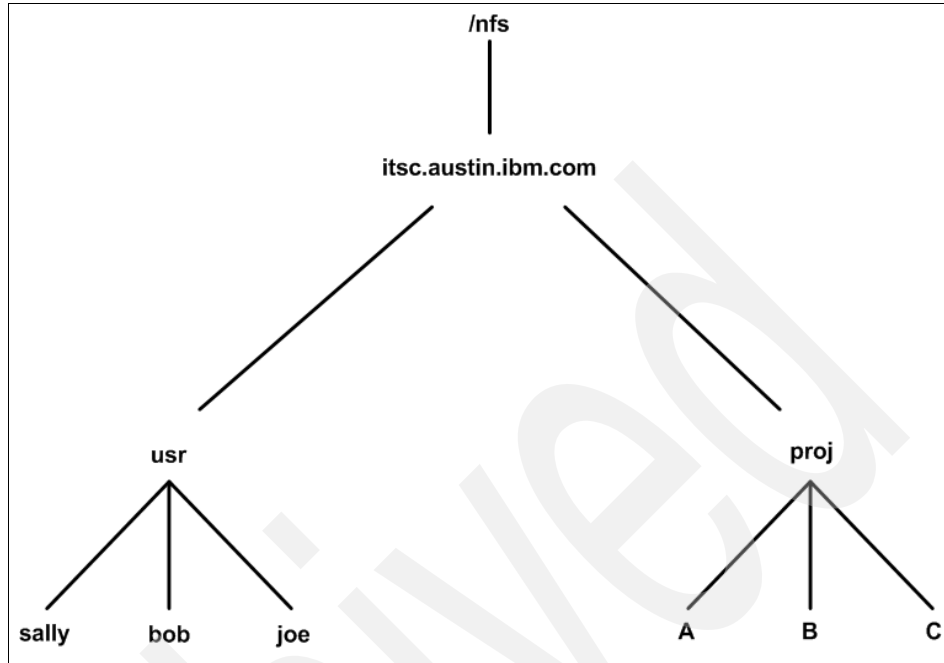


Figure 5-2 Pseudo file system: Client view

As the previous example shows, NFS creates an extension of the local file system. The same hierarchical structure is visible to the user.

5.4 NFSv4 ACLs

NFSv4 ACLs are similar to Microsoft Windows NTFS ACLs, but they are not identical. The developers of the NFSv4 standard chose the Windows ACLs model over the POSIX standard because the Windows ACL model is both richer and more widely deployed. Although many UNIX vendors implemented ACLs based on the POSIX Draft ACL specification, those implementations tended to be proprietary, and the POSIX specification was never standardized.¹

¹ From the paper *The NFS Version 4 Protocol*, Pawlowski, B. et al., available at: <http://www.nluug.nl/events/sane2000/papers/pawlowski.pdf>

Note: In order to use NFSv4 ACLs, the server file system must support them. As of this writing, AIX 5L Version 5.3 only supports NFSv4 ACLs in two file system types: enhanced journaled file system (JFS2) with the extended attribute format set to version 2 (EAv2), and General Parallel File System (GPFS). You can migrate from EAv1 to EAv2 using the **chfs -a ea=v2** command. For more information about NFSv4 ACL support, see *AIX 5L Version 5.3 Security Guide*, SC23-4907, and *AIX 5L Differences Guide Version 5.3 Edition*, SG24-7463.

NFSv4 ACL format

According to the NFSv4 protocol specification, an NFSv4 ACL is an array of access control entries (ACEs) composed of four elements: a type, a set of flags, an access bit mask, and an identity. As implemented in the AIX 5L JFS2 EAv2 file system, each ACL can be a maximum of 64 KB.

The textual representation of an NFSv4 ACL consists of a list of access control entries (ACEs), one ACE per line. Each ACE has four elements in the following format:

IDENTITY ACE_TYPE ACE_MASK ACE_FLAGS

IDENTITY has the format:

IDENTITY_type:(IDENTITY_name or IDENTITY_ID or IDENTITY_who):

Table 5-1 lists possible values for IDENTITY_type.

Table 5-1 ACE IDENTITY_type values

IDENTITY_type	Description
u	User (IDENTITY_name or IDENTITY_ID)
g	Group (IDENTITY_name or IDENTITY_ID)
s	Special who string (IDENTITY_who)

IDENTITY_name is the user or group name.

IDENTITY_ID is the user or group numeric ID.

IDENTITY_who is a special who string that needs to be understood universally rather than in the context of a particular NFS domain. Table 5-2 on page 92 shows possible IDENTITY_who strings.

Table 5-2 ACE special who strings supported by AIX 5L

IDENTITY_who	Description
OWNER@	The owner of the file
GROUP@	The group associated with the file
EVERYONE@	The world

The additional special who strings, shown in Table 5-3, specified in RFC 3530 are not currently supported in AIX 5L.

Table 5-3 ACE special who strings not supported by AIX 5L

IDENTITY_who	Description
ANONYMOUS@	Accessed without any authentication
AUTHENTICATED@	Any authenticated user (opposite of ANONYMOUS)
INTERACTIVE@	Accessed from an interactive session
NETWORK@	Accessed through the network
DIALUP@	Accessed through a dialup connection
BATCH@	Accessed from a batch job
SERVICE@	Accessed from a system service

ACE_TYPE is a single character. Table 5-4 shows possible values.

Table 5-4 ACE_TYPE values

ACE_TYPE	Description
a	Allow access
d	Deny access
l	Generate a system alarm when an access is attempted (currently not supported in AIX 5L)
u	Generate an audit log entry when an access is attempted (currently not supported in AIX 5L)

ACE_MASK is a set of permission flags (that is, permission bits) that can be combined together without any separator. Table 5-5 on page 93 shows the possible flags. Note that the flag values are case-sensitive.

Table 5-5 *ACE_MASK* values

ACE_MASK	RFC 3530 name	Description
r	READ_DATA or LIST_DIRECTORY	Permission to read the data of the file or list the contents of the directory
w	WRITE_DATA or ADD_FILE	Permission to modify the file's data or add a new file to the directory
p	APPEND_DATA or ADD_SUBDIRECTORY	Permission to append data to the file or add a new subdirectory to the directory
R	READ_NAMED_ATTRS	Permission to read the named attributes of the file or directory
W	WRITE_NAMED_ATTRS	Permission to write the named attributes of the file or directory
x	EXECUTE	Permission to execute the file or traverse the directory
D	DELETE_CHILD	Permission to delete files or subdirectories from within the directory
a	READ_ATTRIBUTES	Permission to read basic attributes (non-ACLs) of the file or directory
A	WRITE_ATTRIBUTES	Permission to change basic attributes (non-ACLs) of the file or directory
d	DELETE	Permission to delete the file or directory
c	READ_ACL	Permission to read the ACL of the file or directory
C	WRITE_ACL	Permission to change the ACL of the file or directory
o	WRITE_OWNER	Permission to change the owner of the file or directory
s	SYNCHRONIZE	Permission to access file locally at the server with synchronous reads and writes

ACE_FLAGS (optional) is a combination of one or more of two-letter flags without any separator, as shown in Table 5-6 on page 94. Four of the currently defined flags have to do with ACL inheritance, and the other two have to do with auditing. The inheritance flags only have meaning when applied to a directory. The auditing flags only have meaning when used with the audit or alarm ACE types.

Table 5-6 ACE_FLAG inheritance-related values

ACE_FLAG	RFC 3530 name	Description
fi	FILE_INHERIT	Indicates that this ACE should be added to each newly created non-directory file.
di	DIRECTORY_INHERIT	Indicates that this ACE should be added to each newly created subdirectory.
oi	INHERIT_ONLY	Indicates that this ACE does not apply to the current directory; it is only to be added to newly created files/subdirectories as specified by the previous two flags.
ni	NO_PROPAGATE_INHERIT	Indicates that this ACE should be added to newly created files/subdirectories immediately under the directory, but subdirectories should not pass it on to their children.

Table 5-7 ACE_FLAG auditing-related values

ACE_FLAG	RFC 3530 name	Description
sf	SUCCESSFUL_ACCESS_ACE_FLAG	Generate audit or alarm when an access attempt succeeds
ff	FAILED_ACCESS_ACE_FLAG	Generate audit or alarm when an access attempt fails

NFSv4 ACL permission restrictions

Some permission bits are interrelated and must be used together under the following circumstances. The *WRITE_DATA* (w) and *APPEND_DATA* (p) bits must be specified together in a file's ACE, or in a directory's ACE that has the FILE_INHERIT flag set.

Special user permissions

In NFSv4, there are two classes of users that have special access to files:

- ▶ The UNIX super user (UID=0) is allowed all access permissions, regardless of the ACE permission bit settings. (The only exception to this is execute permission.) This special access applies to processes running on the NFS server, and processes running on NFS clients that have been given root access with the **exportfs** command.
- ▶ The owner of a file always has the permissions READ_ACL, WRITE_ACL, READ_ATTRIBUTES, and WRITE_ATTRIBUTES, regardless of the actual settings in the ACL.

5.4.1 NFSv4 ACLs: ACL evaluation

In order to properly use NFSv4 ACLs, it is important to understand how they are evaluated when determining whether an access request will be granted or denied.

Per the RFC 3530 NFSv4 standard and the *AIX 5L Version 5.3 Security Guide*, SC23-4907, an AIX 5L NFSv4 server evaluates the ACL list from the top down, applying the following rules:

- ▶ Only ACEs that have a who that matches the requester are considered. The credentials of the requester are not checked while processing the ACE with special who EVERYONE@.
- ▶ Each ACE is processed until all of the bits of the requester's access have been allowed, or at least one of the requested bits not previously allowed has been denied.
- ▶ After a permission bit has been allowed, it is no longer considered in the processing of later ACEs.
- ▶ If a deny ACE_TYPE is encountered where the ACE_MASK has bits in common with not yet allowed bits in the request, access is denied, and the remaining ACEs are not processed.
- ▶ If the entire ACL has been processed and some of the requested access bits still have not been allowed, access is denied.

NFSv4 ACL evaluation examples

The following examples help illustrate ACL evaluation. For more examples, see *AIX 5L Version 5.3 Security Guide*, SC23-4907. See Example 5-9.

Example 5-9 ACL on a file for user Sally

```
*
* ACL_type   NFS4
*
*
* Owner: sally
* Group: staff
*
g:sales:      d      wp
s:(OWNER@):   a      rRWDaAdcCs
s:(OWNER@):   d      wpo
s:(GROUP@):   a      rwpRxadcs
s:(GROUP@):   d      WDACo
s:(EVERYONE@): a      rwpRxadcs
s:(EVERYONE@): d      WDACo
```

Given the ACL shown in Example 5-9 on page 95 on a file, if the user Sally requests READ_DATA (r) and WRITE_DATA (w) access, the ACL evaluation proceeds as follows:

1. The “s:(OWNER@):a...” ACE is processed because Sally owns the file.
 - READ_DATA is allowed because that bit is set in the ACE_MASK.
 - WRITE_DATA is not yet allowed because it is not set in the ACE_MASK.
2. The “s:(OWNER@):d...” ACE is processed because Sally owns the file.
 - WRITE_DATA is denied because that bit is set in the ACE_MASK and WRITE_DATA has not yet been allowed by a previous ACE.
3. No further ACEs are processed, and the requested access is denied.

Notes:

- ▶ In the previous example, even though the GROUP@ and EVERYONE@ ACEs allow WRITE_DATA access, Sally is denied WRITE_DATA access because it is specifically denied by the owner ACE.
- ▶ The ACE order is important. If the group allow ACE had appeared in the list before the owner deny ACE, Sally would be allowed write access to the file.

If the user Sally, who is a member of the group staff, requests READ_DATA (r) and EXECUTE (x) access, the ACL evaluation proceeds as follows:

1. The “s:(OWNER@):a...” ACE is processed because Sally owns the file.
 - READ_DATA is allowed because that bit is set in the ACE_MASK.
 - EXECUTE is not yet allowed because it is not set in the ACE_MASK.
2. The “s:(OWNER@):d...” ACE is processed because Sally owns the file.
 - EXECUTE is not yet denied because it is not set in the ACE_MASK.
3. The “s:(GROUP@):a...” ACE is processed because Sally is a member of the group staff, which owns the file.
 - EXECUTE is allowed because that bit is set in the ACE_MASK.
4. All requested permission bits have now been allowed. No further ACLs are processed, and the requested access is granted.

If the user Joe, who is a member of the group sales, requests READ_DATA (r) and WRITE_DATA (w) access, the ACL evaluation proceeds as follows:

1. The “g:sales:d...” ACE is processed because Joe is a member of the group sales.
 - READ_DATA is not denied because it is not set in the ACE_MASK.

- WRITE_DATA is denied because that bit is set in the ACE_MASK and WRITE_DATA has not yet been allowed by a previous ACE.
2. No further ACEs are processed, and the requested access is denied.

If Joe requests just READ_DATA (r) access, the ACL evaluation proceeds as follows:

1. The “g:sales:d...” ACE is processed because Joe is a member of the group sales.
 - READ_DATA is not denied because it is not set in the ACE_MASK.
2. The “s:(EVERYONE@):a...” ACE is processed.
 - READ_DATA is allowed because it is set in the ACE_MASK.
3. All requested permission bits have now been allowed. No further ACLs are processed, and the requested access is granted.

Relationship between NFSv4 ACLs and UNIX permissions

Familiarity with standard UNIX read (r), write (w), and execute (x) permission bits often does not directly correlate with the myriad of bits found in the ACE_MASK. However, as can be seen from the previous ACE_MASK definitions, the r, w, and x permission bits basically provide the same access as the standard UNIX permission bits.

For example, as with UNIX permissions, the w permission bit in a directory’s ACL affects the user’s ability to create, delete, and rename files and subdirectories within that directory, rather than changing the contents of those files and subdirectories.

The difference comes into play when mapping the rwx bits to user (owner), group, and other. This mapping is unspecified in RFC 3530. Example 5-10 shows the mapping observed in AIX 5L.

Example 5-10 Mapping rwx bits to user, group, and other

```
*
* ACL_type   NFS4
*
*
* Owner: sally
* Group: staff
*
s:(OWNER@):  a      cCs
s:(OWNER@):  d      o
s:(GROUP@):  a      rRxadcs
s:(GROUP@):  d      wpWDACo
s:(EVERYONE@): a      rwpRxadcs
s:(EVERYONE@): d      WDACo
```

Given a file with the following ACL, running the `ls -l` command on the file shows:

```
-rwxr-xrwx  1 sally  staff          0 Jul 30 11:20 testfile
```

One might conclude that the bits will map directly from OWNER@ to user, GROUP@ to group, and EVERYONE@ to other. As can be seen in the previous example, this is not the case. It becomes evident that the user permissions show as `rwx`, while the `s:OWNER::a` ACE has none of those bits set. Furthermore, even though the `ls -l` output makes it look like the user Sally has write access to the file, she actually does not. Evaluating the ACEs from top down, write access is denied by the `s:GROUP::d` entry.

Based on this, the conclusion must be drawn that the standard UNIX permissions bits cannot be used to reliably predict access when using NFSv4 ACLs.

5.4.2 NFSv4 ACLs: Administration

In AIX 5L, NFSv4 ACLs (and AIXC ACLs) can be administered from either the NFS server or an NFSv4 client through the command line or through the AIX 5L Web-based System Manager.

Manipulating ACLs through the command line

ACLs can be administered using the following commands:

aclget	Writes the textual representation of an ACL to standard output or to a named file.
aclput	Replaces the contents of an ACL from a textual representation provided either from standard input or from a named file.
acledit	Retrieves the ACL's textual representation into a text editor (specified by the EDITOR environment variable) and then replaces the ACL from the modified text.
aclconvert	Converts an ACL's format from either AIXC to NFS4 or from NFS4 to AIXC. The translation is not necessarily straightforward. Use this with caution, and make sure that the end result is what you intended.
aclgettypes	Returns a list of the ACL types supported by the file system that contains a given file or directory.

Note: The **aclconvert** and **aclgettypes** commands are new in AIX 5L Version 5.3.

Manipulating ACLs through the Web-based System Manager

The AIX 5L Web-based System Manager can also be used to manipulate ACLs. This might be easier for novice users to grasp because it is GUI-based. However, Web-based System Manager might only be useful for only the most basic operations, and experienced administrators are likely to rely on the command line utilities.

Here is an example of how to use Web-based System Manager to change an ACL:

1. Start the Web-based System Manager console and double-click the **File Systems** icon. Then, double-click the **Overview and Tasks** icon. A window similar to that shown in Figure 5-3 opens.
2. Select **Access Control List** either from the main window or from the Filesystems menu, as shown in Figure 5-3.

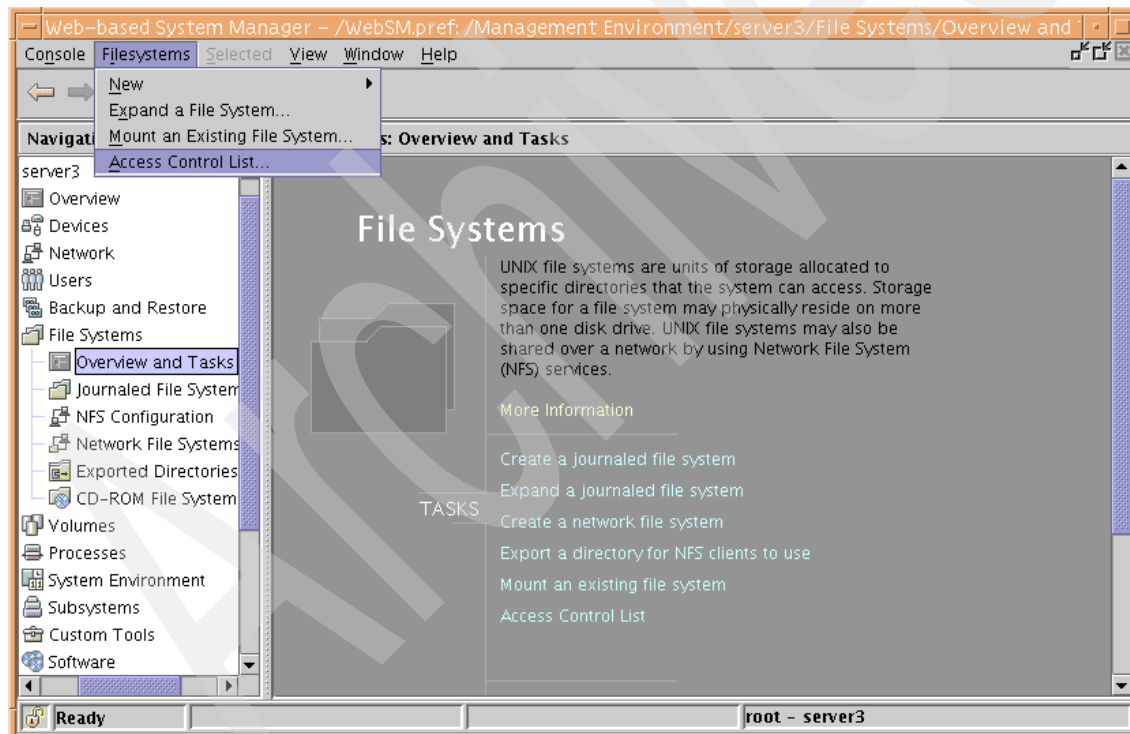


Figure 5-3 Web-based System Manager File Systems: Overview and Tasks window

3. Either type in the full path name of the file or directory of the ACL that requires modification or click **Browse** to choose the file or directory from the GUI. After entering the name, either type Enter or click **Next**. See Figure 5-4.

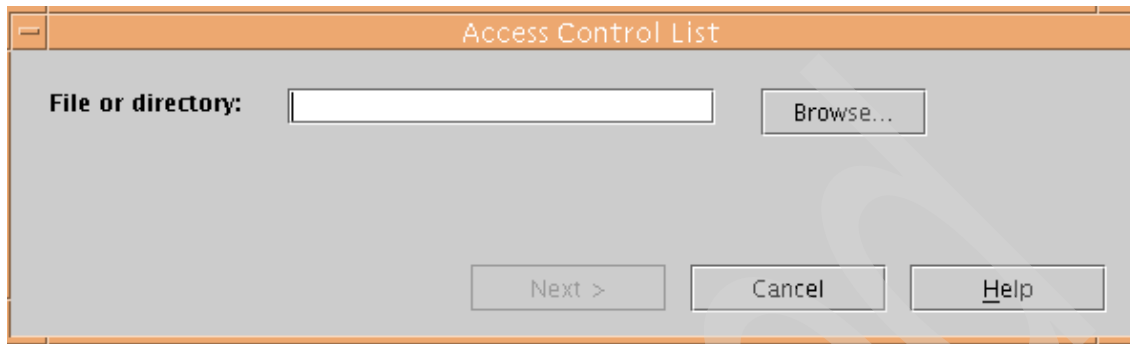


Figure 5-4 Web-based System Manager ACL File or directory name prompt

4. Make sure that **Edit ACL** is selected and click **Next**, as shown in Figure 5-5.

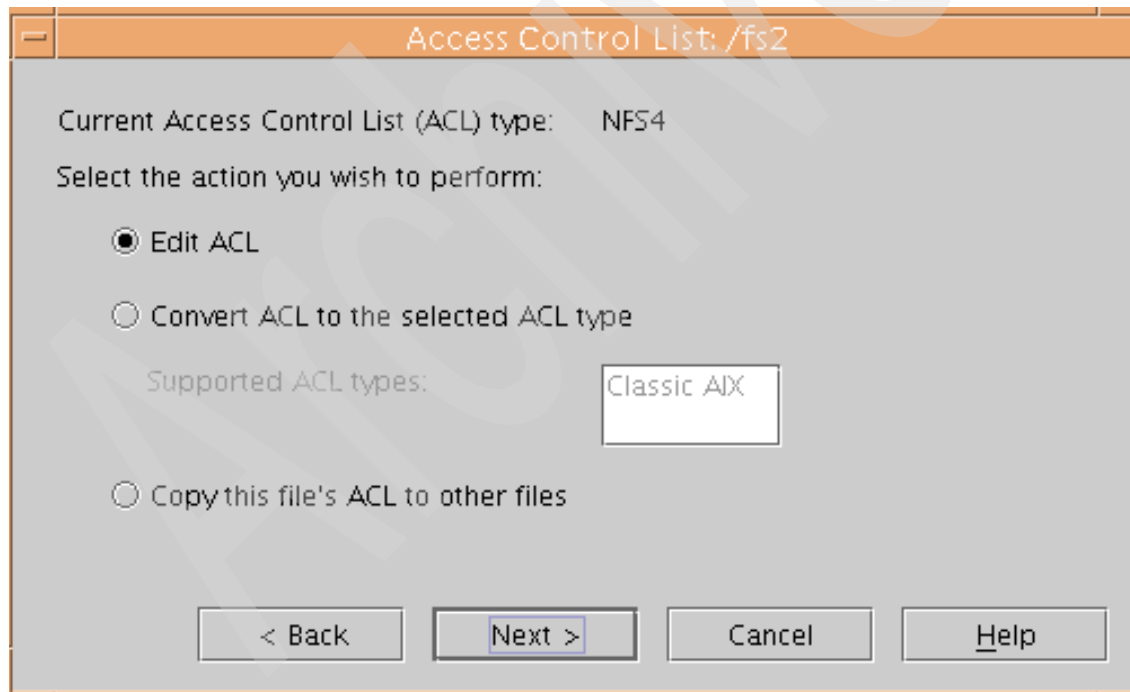


Figure 5-5 Web-based System Manager ACL operation selection

5. Select the ACE you want to change and click **Edit**, as shown in Figure 5-6. The ensuing window has two tabs: General and Access Mask. Under the General tab, the user type and identity, the ACE type, and the ACE flags (inheritance, audit and alarm) can be set. If the Access Mask tab is selected, a window similar to the one shown in Figure 5-7 on page 102 opens.

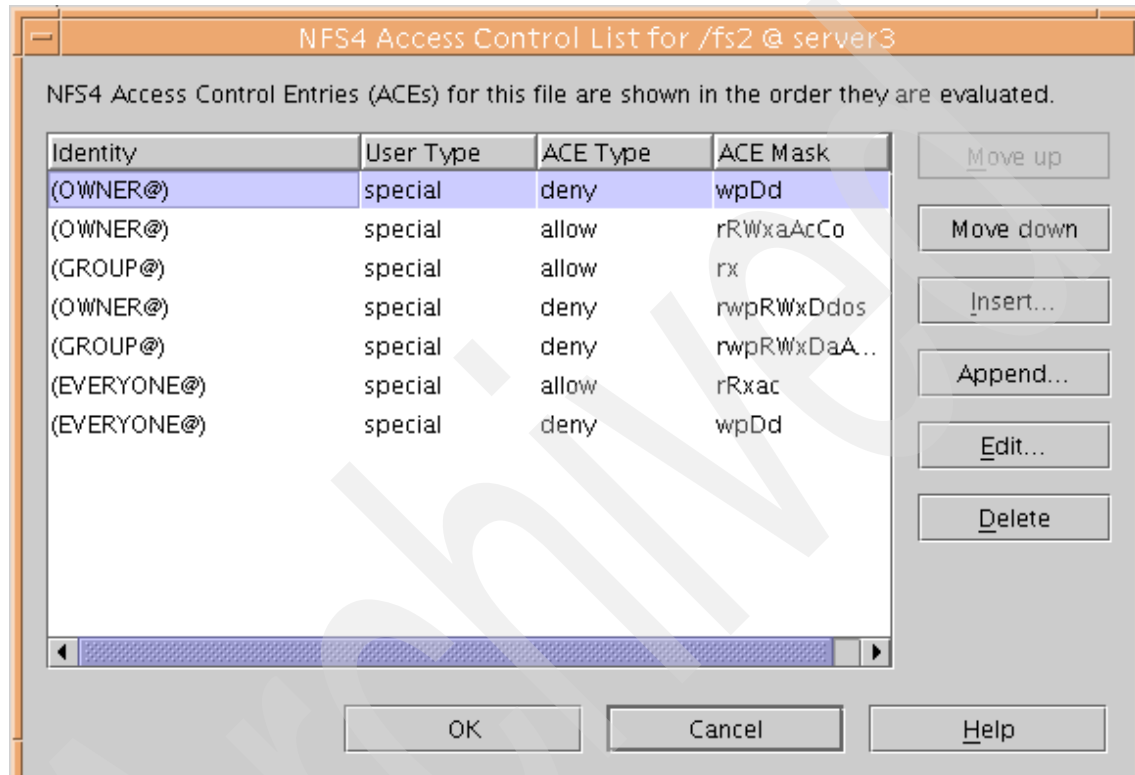


Figure 5-6 Web-based System Manager ACL edit window

6. Ensure that the selected access mask entries are correct and click **OK** to return to the ACL edit window (Figure 5-7).

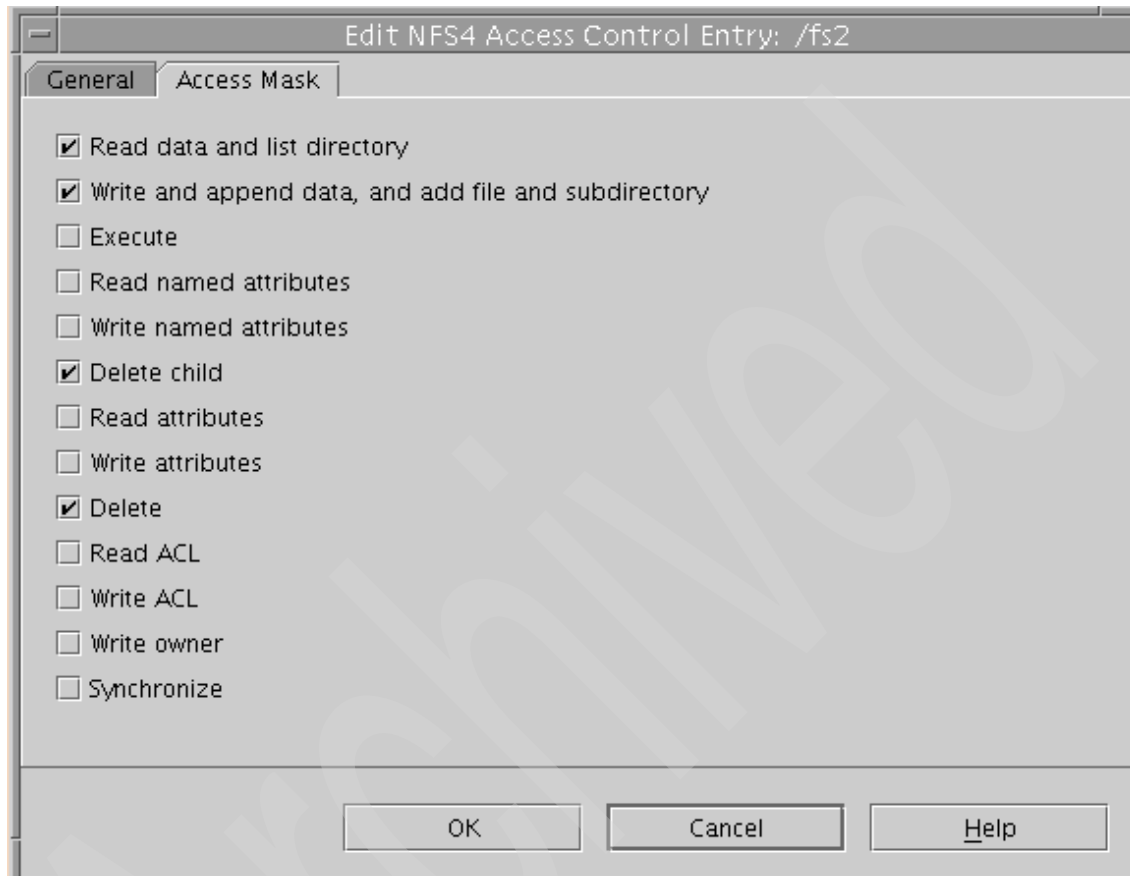


Figure 5-7 Web-based System Manager ACE mask window

7. Repeat steps 5 and 6 for other ACEs that require modification. When the process is complete, click **OK** on the ACL edit window to display a pop-up window indicating the status of the operation. Click **Close** on that pop-up window after reviewing the results for accuracy.

This concludes the Web-based System Manager example.

Using the **chmod** command

When working with files and directories that use NFSv4 ACLs, the **chmod** command can only be used to set UNIX permission bits other than those stored in the ACL. These bits are the **setuid/setgid** and **sticky** bit. Table 5-8 on page 103 shows **chmod** command operations that are compatible with NFSv4 ACLs.

Table 5-8 *chmod operations compatible with NFSv4 ACLs*

chmod command	Description
chmod u+s chmod u-s	Set or unset the setuid bit
chmod g+s chmod g-s	Set or unset the setgid bit
chmod +t chmod -t	Set or unset the sticky bit

Important: Using the **chmod** command to manipulate the rwx permission bits, either in octal form (for example, 755) or in symbolic form (for example, u+x) will replace the NFSv4 ACL with an AIXC ACL, wiping out the original permissions that were on the file/directory.

Never use the octal form of the **chmod** command if NFSv4 ACLs are in use. Even if the rwx bits are not specified in the **chmod** command, use of the octal form will replace the NFSv4 ACL with an AIXC ACL.

Note: If the **chmod** command is used to manipulate rwx permission bits on an NFS client and, again on the client, **aclget** is subsequently executed against the file, the ACL will still appear to be an NFSv4 ACL. It will, however, be an AIXC ACL on the NFSv4 server. The NFSv4 protocol translates AIXC ACLs that have extended permissions disabled to look like NFSv4 ACLs at the client.

5.4.3 NFSv4 ACLs: ACL inheritance and umask

Does the UNIX umask have any impact on inherited ACL settings when creating a new file or directory? The answer is no. The umask has no effect on inherited ACL settings when using NFSv4 ACLs.

For example, if a directory has the ACL shown in Example 5-11, a file created in that directory will have the same ACL, even if the umask is set to 777.

Example 5-11 Example ACL

```
*
* ACL_type   NFS4
*
*
* Owner: root
```

```

* Group: system
*
s:(OWNER@): a      rwpRWxDaAdcCs  fidi
s:(OWNER@): d      o      fidi
s:(GROUP@): a      rwpRxadcs    fidi
s:(GROUP@): d      WDACo    fidi
s:(EVERYONE@): a    rwpRxadcs    fidi
s:(EVERYONE@): d    WDACo    fidi

```

ACL inheritance and move versus copy

Table 5-9 describes the impact that the UNIX **mv**, **cp**, and **cp -p** commands have on a file's ACL (provided the destination file system also supports NFSv4 ACLs).

Table 5-9 UNIX commands and impact on ACLs

Command	Resulting file ACL
mv	The file retains the same ACL it had in the original location if the source and target file systems are the same. If not, ACL assignment occurs as per the cp command.
cp	The file inherits its ACL from the directory where it is being placed, just as though it were a newly created file.
cp -p	The file retains the same ACL it had in the original location.

Directory structure and ACLs

Numerous options are available when organizing data into a directory structure and implementing ACLs to control access to that data. For example, a site might choose either of two methods to control read access to data:

- ▶ **Method 1:** Controlling access at the directory, or container level, by maintaining uniform access permissions on files and subdirectories within a directory.
- ▶ **Method 2:** Leaving access wide open at the directory level and setting unique access restrictions at each individual file.

Depending on the site's requirements, the first or second method might make more sense. Each method has its own characteristics. We describe some of these characteristics.

Characteristics of method 1:

- ▶ It is easier for most users to keep track of permissions when files with like permissions are grouped together.
- ▶ Permissions can be easily changed though a bulk replacement of the ACLs. (See "Maintaining an existing directory structure" on page 107.)

- ▶ If a subset of files in a directory needs to have their permissions changed, the files must be moved to a different directory. Directory location changes can be disruptive to operations. (For example, symbolic links and other path name references might need to be updated.)
- ▶ A file could inadvertently inherit incorrect permissions if it is placed in the wrong directory.
- ▶ If a directory has less restrictive access permissions than a parent directory, an NFS client might possibly mount that directory on a path that has more open access. A user whose access would normally be blocked by the parent directory might then be able to access the directory through the mounted path.

Characteristics of method 2:

- ▶ Every file's permissions can be tailored to its unique access requirements.
- ▶ A file does not need to be moved when its permissions need to be different from the files around it.
- ▶ It is more difficult to keep track of the different file permissions.
- ▶ It is easy to mistakenly overwrite a file's permissions through a bulk update, and it is relatively complicated, first to detect a mistake, and then to restore the correct permissions after a mistake has happened. (You need to know through some external source what the original permissions were.)
- ▶ To prevent inadvertent access, create each file with the most restrictive set of permissions (through inheritance), requiring manual intervention by the user to share that file with others.

Maximizing the benefits of ACL inheritance

If you choose method 1 for implementation, organize the directory structure to maximize the use of ACL inheritance. To do this, carefully plan the directory structure so that files and subdirectories with the same access requirements are collocated under a single parent directory.

The following example helps illustrate this concept.

An organization is composed of three departments: engineering (eng), sales, and human resources (hr). Users from each of the departments are working on two different projects: projA and projB. For business reasons, the two projects must be entirely separate, and users working on one project must not be able to access data belonging to the other project.

Each department has its own directory for data, and each creates separate project directories under its directory. Figure 5-8 on page 106 depicts the resulting structure.

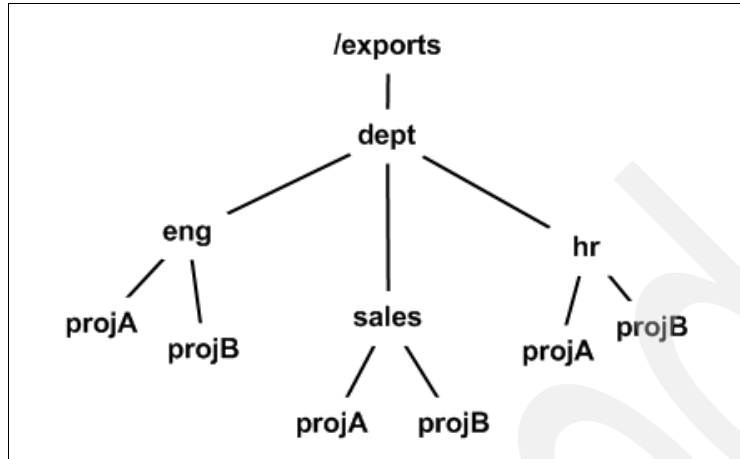


Figure 5-8 Directory structure that makes poor use of ACL inheritance

This structure has the project directories replicated under each department. If a permissions change is required for one of the projects, the same changes must be made in three different places.

Figure 5-9 depicts a directory structure that better lends itself to managing the project permissions.

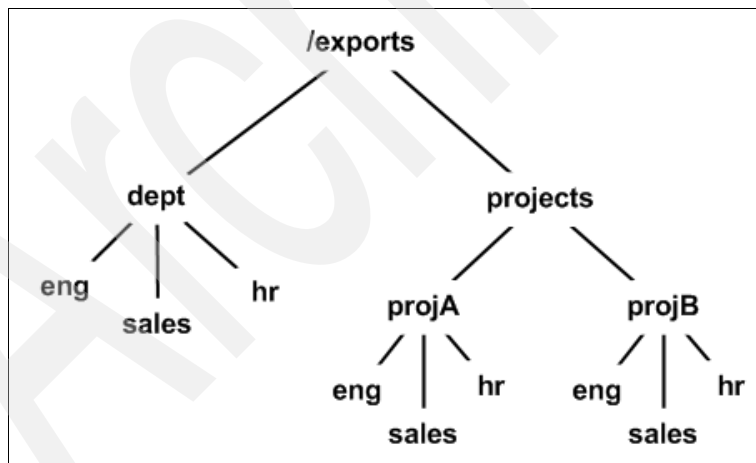


Figure 5-9 Directory structure that makes better use of ACL inheritance

This structure has a separate projects directory where the permissions for each special project can be managed in one place. The departments still maintain non-project-related data under the dept directory.

Maintaining an existing directory structure

Inheritance manages the setting of permissions on newly created files and directories, but it does not affect permissions for existing files. No amount of forward planning will eliminate the need to eventually change permissions on an existing directory structure and all the files it contains.

There are two possible ways to make a large-scale permissions change:

- ▶ Method 1: Make the change to one file or directory and then propagate the change to other files or directories by copying the whole ACL from the file or directory that was previously changed.
- ▶ Method 2: Incrementally change the ACL for every file or directory.

The first method is simpler to implement, but all files or directories being changed will acquire exactly the same permissions, eradicating any variations that might have existed. This might be a good thing or a bad thing, depending on how the directory structure has been arranged. The second method is much more complex to implement, but it allows for other differences to exist in the ACLs. This is another example where carefully planning the initial directory structure around permissions requirements can make administration easier.

The rest of this section illustrates possible ways to implement the ACL propagation method 1.

It is possible to propagate an ACL to an entire directory structure using a combination of the **aclget** and **aclput** commands as follows:

```
aclget dirname | aclput -R dirname
```

Different source and directory names can be specified, or the same directory name for both source and destination can be used to copy a directory's ACL to all its descendants (including itself).

Important: Only use the **aclput -R** command on a directory structure that has a uniform permissions structure. The command applies a wholesale replacement of all existing ACLs at and below the specified directory. Any variations in ACLs that previously existed will be lost.

Using the **aclget** | **aclput** command combination is convenient, but some drawbacks exist:

- ▶ If the name of the source directory is mistyped, existing permissions in the destination directory are completely destroyed. This can be quickly remedied by reissuing the command with the correct source name, but in the meantime, access will be blocked to any user or application attempting to access the data.

- The **aclput -R** command stops at the first error encountered, leaving the remaining files untouched.

The sample script shown in Example 5-12 addresses both these issues. It does not attempt to run **aclput** if either the source or destination does not exist, and it runs **aclput** on each individual file or directory so that all possible ACL changes are made.

Example 5-12 Sample script for copying an ACL (with recursive option)

```
#!/usr/bin/ksh
#
# copy_acl.sh
#
# Copy the ACL for the given source file/directory to other files/directories
#

# Name of this script
scrname=${0##*/}

#
# Functions
#

function usage {
    echo "Usage: $scrname [-R] <source> <dest>"
    echo "  where"
    echo "    -R indicates a recursive copy"
    echo "        (copy ACL to all files and directories below and including"
    echo "        the destination.)"
    echo "  <source> = the name of the file or directory to copy the ACL from"
    echo "  <dest>   = the name of the file or directory to copy the ACL to"

    exit 1
}

if [[ $# -eq 0 ]]
then
    usage
fi

#
# Process input parameters
#

if [[ "$1" = "-R" ]]; then
    SETSUBTREE="true"
    shift
else
    SETSUBTREE="false"
fi
```



```

if [[ -n "$1" ]]; then
    SRC_NAME="$1"
else
    usage
fi

if [[ -n "$2" ]]; then
    DEST_NAME="$2"
else
    usage
fi

#
# Initialize other variables
#

NBERR=0
TMP_ACLFILE="/tmp/.AIXACL_$$"

if [[ -e "${SRC_NAME}" ]]; then
    aclget -o "${TMP_ACLFILE}" "${SRC_NAME}"
    NBERR=$?
else
    echo "Source \"${SRC_NAME}\" does not exist"
    NBERR=1
fi

if [[ "${NBERR}" -eq 0 ]]; then
    if [[ -e "${DEST_NAME}" ]]; then
        if [[ -d "${DEST_NAME}" && "${SETSUBTREE}" = "true" ]]; then
            find "${DEST_NAME}" -print | while read NAME
            do
                aclput -i "${TMP_ACLFILE}" "${NAME}"
                (( NBERR += $? ))
                ls -dl "${NAME}"
            done
        else
            aclput -i "${TMP_ACLFILE}" "${DEST_NAME}"
            (( NBERR += $? ))
            ls -dl "${DEST_NAME}"
        fi
    else
        echo "Destination \"${DEST_NAME}\" does not exist"
        NBERR=1
    fi
fi

rm -f "${TMP_ACLFILE}"
exit ${NBERR}

```

5.4.4 NFSv4 ACLs: Permissions scenarios

The following scenarios help illustrate how NFSv4 ACLs can be applied:

- ▶ Scenario 1: Restricting home directory access to just the associated user, and not allowing users to change permissions and open up their home directories to others.
- ▶ Scenario 2: Ensuring that a particular group is denied access to a set of data.

ACL scenario 1: Home directories

Users' home directories can be collectors for all types of data. Users often place data in their home directory while working with it. The data might have originated in a different directory with strict access controls, and the home directory's permissions should not permit wider access to that data. One way to manage this is by locking each home directory so that only its associated user can access it.

This is difficult to do with standard UNIX permissions. There are two basic options:

- ▶ Make the user the owner of the directory and allow only owner access.
Because the user owns the directory, the user can change its permissions, which is not the desired behavior.
- ▶ Create a group for each user, where the user is the only member, make the home directory owned by root and the user's group, and allow only owner and group access.

The user cannot change the directory permissions, but this option requires maintaining a whole set of groups, one for each user.

This is easier to do with NFSv4 ACLs. Make root the owner of the directory and add a user ACE to allow the user access to the directory. Example 5-13 on page 111 shows how that ACL would look.

Example 5-13 ACL to make root the owner and add a use ACE to allow the user access

```
*
* ACL_type   NFS4
*
* Owner: root
* Group: system
*
s:(OWNER@):   a      rwpRWxDaAdcCs  fidi
s:(OWNER@):   d      o      fidi
u:sally(sally@nfsdom1): a      rwpRWxDaAdcs
u:sally(sally@nfsdom1): d      Co
s:(GROUP@):   d      rwpRWxDaAdcCos  fidi
s:(EVERYONE@): d      rwpRWxDaAdcCos  fidi
```

Tip: The user ACEs do not need to be inherited because files created below the directory will be owned by the user.

The user can open permissions for files or subdirectories that the user creates in the directory because the user owns them, but the home directory itself will still block access to those files.

Note: It might be possible to NFS mount a lower level directory that has more open permissions and gain access to those files, but normally, the mount operation is under system administrator control. If mounts are managed correctly, users will not be able to get directly at lower directories under the home directory.

ACL scenario 2: Block a group's access

If you have two subcontractors working on a project, and you want to make sure that the subcontractors are not able to access each other's data, you use the scenario discussed in this section.

Create a group for each subcontractor, put each subcontractor's data in a separate directory structure, and put an ACE at the top of the ACL that denies access to the other subcontractor. (It is important that the ACE be at the top of the list to prevent other ACEs from allowing access before the subcontractor's access is blocked.)

Given that the groups are company1 and company2, the ACL on company1's data would look like that shown in Example 5-14 on page 112.

Example 5-14 ACL to deny company2 access to company1's data

```
*
* ACL_type   NFS4
*
*
* Owner: root
* Group: system
*
g:company2(company2@nfsdom1):      d      rwpRWxDaAdcCos  fidi
s:(OWNER@):      a      rwpRWxDaAdcCs   fidi
s:(OWNER@):      d      o      fidi
s:(GROUP@):      a      rRxadcS  fidi
s:(GROUP@):      d      wpWDACo fidi
s:(EVERYONE@):   a      rRxadcS  fidi
s:(EVERYONE@):   d      wpWDACo fidi
```

No matter what the rest of the ACEs are, company2 will be denied access to company1's data.

5.4.5 NFSv4 ACLs: ACL evaluation flowchart for NFSv4

The chart in Figure 5-10 on page 113 illustrates the decision making flow of the NFSv4 ACLs.

Note: This chart does not include *ACE special who strings* that are not supported by AIX 5L, but the decision making process is the same.

The chart begins with the user's requested permissions and continues until either:

- ▶ All requested permissions have been found on processed ACL entries.
- ▶ A requested permission is explicitly denied by an ACL entry for this user.

Note: After all of the permissions requested are met by an ACL entry or entries, or after a requested permission is explicitly denied by an ACL entry or entries, the processing halts. No further ACL checking is performed. This means that subsequent ACLs in the list, which might have explicitly provided the user with requested permissions, would never be processed and the user would be denied access to the file or directory.

Users that have never worked with Microsoft Windows NT® style ACLs may find this “looping” concept confusing until they have worked with a few examples and seen how the ACLs are processed. Comparing this chart with the DFS flowchart shown in Figure 10-3 on page 239 can illustrate the contrast as well.

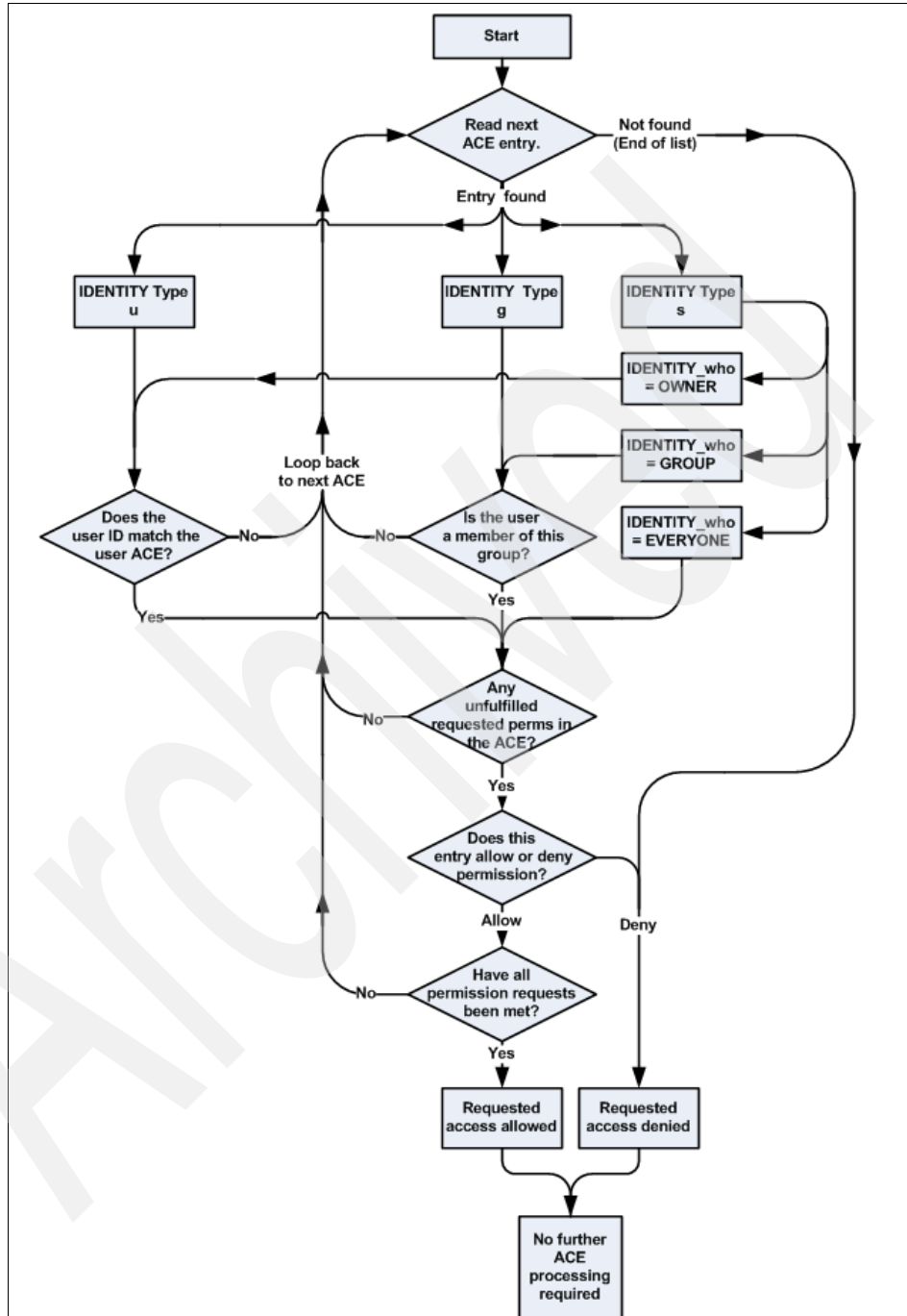


Figure 5-10 NFSv4 ACL decision flow

5.4.6 NFSv4 ACLs: NFSv3 clients

It is possible to make an NFSv3 mount of a file system that contains NFSv4 ACLs. This combination exhibits the following behavior:

- ▶ The NFS server will still grant or deny data access based on the NFSv4 ACLs.
- ▶ The NFS client will not be able to view or manipulate the ACLs directly. For example, an **aclget** command on the client returns the error:
`aclget: The system call does not exist on this system.`
- ▶ If the mount is made with the **ac1** option (**noac1** is the default), the NFS client will be able to manipulate AIXC ACLs, but not NFSv4 ACLs.

Important: It is OK to have NFSv3 clients mount file systems that use NFSv4 ACLs. ACL inheritance and evaluation will work normally on the server. Do not, however, attempt to manipulate access permissions directly from the NFSv3 client. Any permissions change at the NFSv3 client will overwrite the NFSv4 ACL with an AIXC ACL.

Unfortunately, there is no good way to block a user on the NFSv3 client from running **chmod** or **aclput** on file or directories that the user owns. You have to publish policy and rely on well-behaved users. (You can completely disable the **chmod** and **aclput** commands on the client, but that also disables them for other client file systems where using those commands is perfectly valid.)

In addition, keep in mind when using NFSv3 clients that the UIDs and GIDs need to match between the server and client.



Part 3

Preparing to use NFSv4

In this part, we introduce the planning, migration, and implementation methodologies that we employed in the environment created for this book.

For detailed descriptions and implementation considerations of the features introduced in the initial release of AIX 5L V5.3, see the IBM Redbook *Securing NFS in AIX: An Introduction to NFS V4 in AIX 5L Version 5.3*, SG24-7204. You can view this book online or download it from the following location:

<http://www.redbooks.ibm.com/abstracts/sg247204.html>

We include the following decision support flowcharts and planning information table to demonstrate the type of information that needs to be gathered and to illustrate the design decision process.

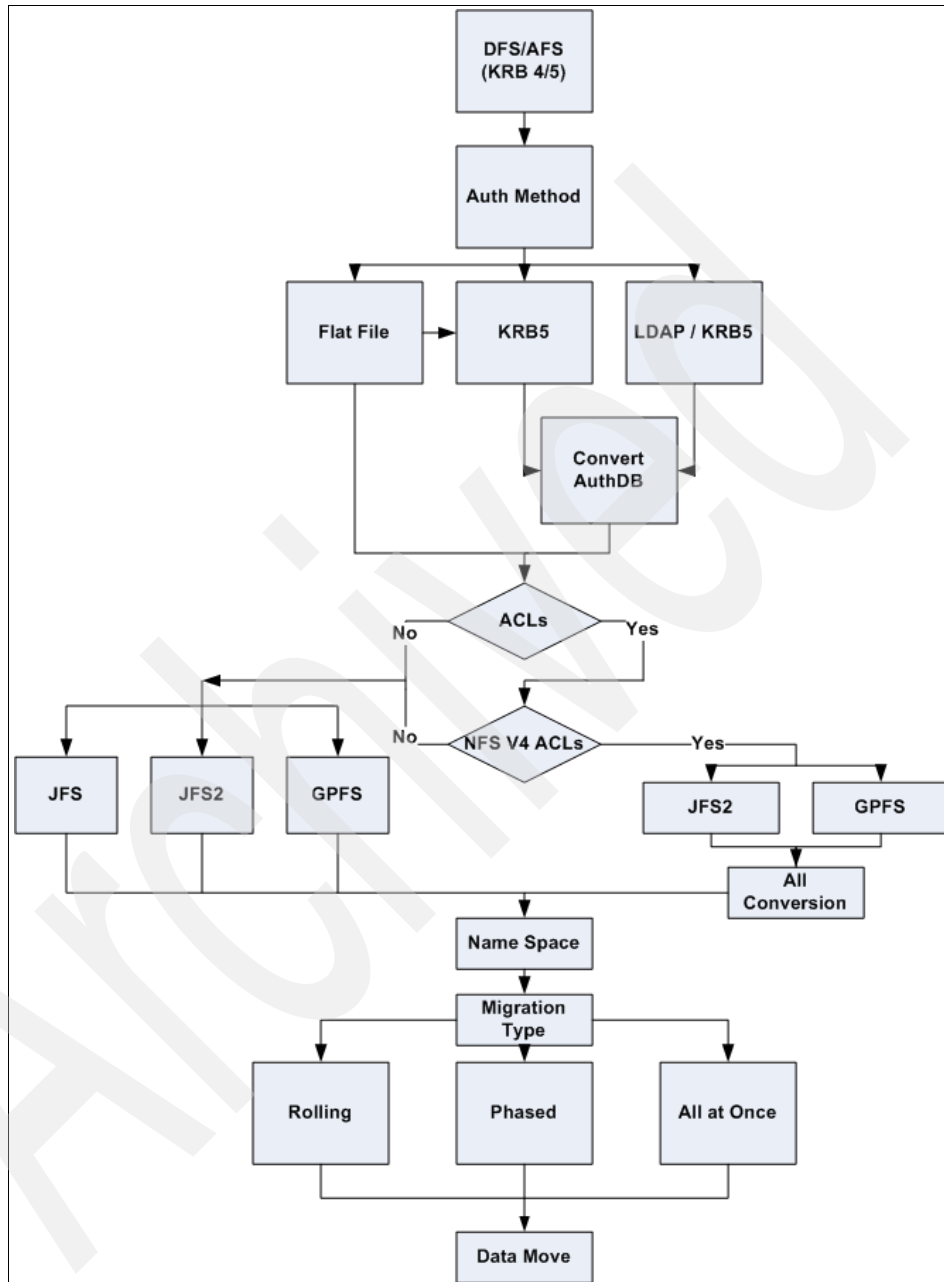


Figure P3-1: Migration decision flowchart

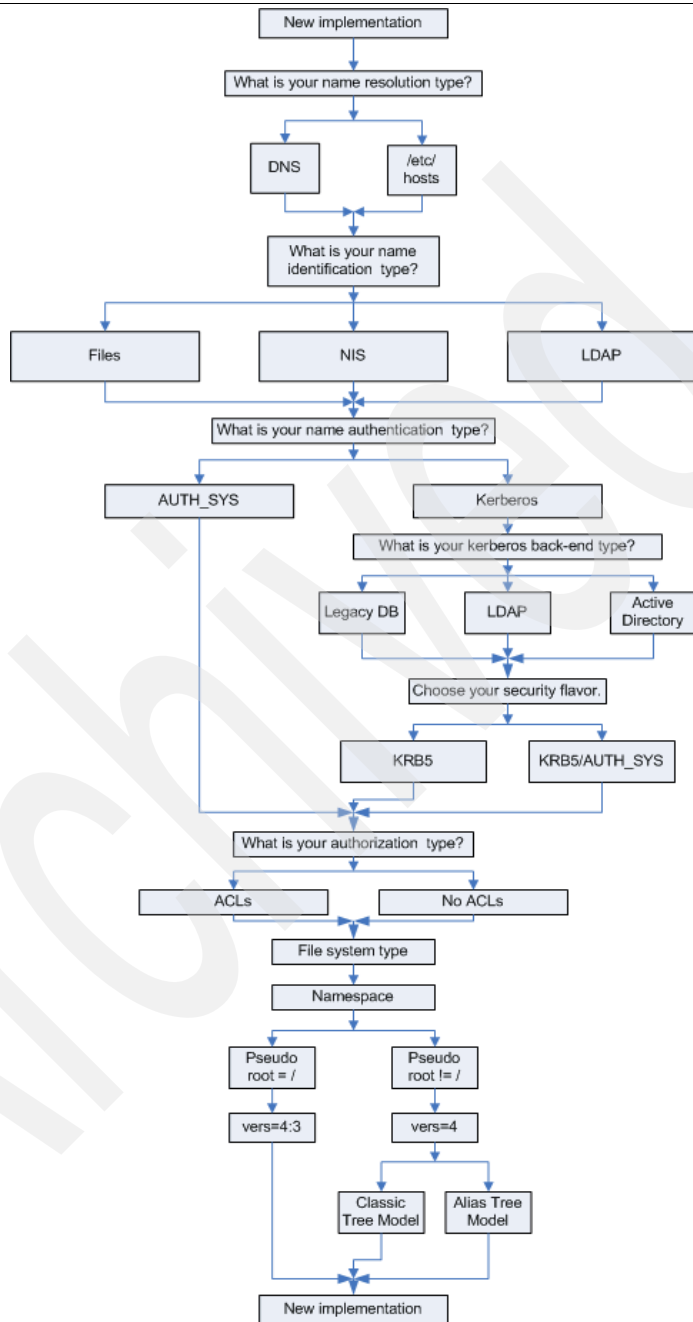


Figure P3-2: New NFSv4 implementation decision flowchart

Table P3-1: Planning worksheet for a new implementation

1. Name resolution type	
2. NFS domain name	
3. Identification method (files, NIS, or LDAP)	
3a. LDAP server name	
3b. LDAP server IP address	
4. Authentication method (AUTH_SYS or Kerberos V5)	
4a. Kerberos realm name	
4b. Kerberos back-end type (established DB, LDAP, Active Directory)	
4c. Kerberos security flavor	
4d. Kerberos server name	
4e. Kerberos server IP address	
5. Authorization method	
6. Underlying file system type (JFS2 or GPFS)	
7. Namespace strategy	
7a. Pseudo root/namespace name	
7b. Classic or alias tree model	

Building an NFSv4 environment

We discuss the following topics in this chapter:

- ▶ Environment used for demonstration scenarios
- ▶ Network Time Protocol (NTP) configuration
- ▶ IBM Tivoli Directory Server V5.2
- ▶ IBM Network Authentication Services (Kerberos V5) server installation
- ▶ IBM Tivoli Directory Server client configuration
- ▶ IBM Network Authentication Services client install and configuration

We also include the following optional steps:

- ▶ Installing GPFS
- ▶ Configuring GPFS

6.1 Environment used for demonstration scenarios

We refer to the following system names and environment during the discussion of the migration and configuration scenarios. All systems use the domain name:

itsc.austin.ibm.com

Therefore, the fully qualified name for *guadalupe* is:

guadalupe.itsc.austin.ibm.com

Table 6-1 shows the systems and functions in the test environment.

Table 6-1 Systems and functions in the test environment

System name	OS version	Function	Additional function
pecos	AIX 5L V5.3 RML01	Kerberos V5 (KRB5)/LDAP server	KRB5: IBM Network Authentication Services LDAP: IBM Tivoli Directory Server
guadalupe	AIX 5L V5.3 RML03	KRB5 client	
sabine	AIX 5L V5.3 RML03	KRB5 client	GPFS node
frio	AIX 5L V5.3 RML03	KRB5 client	GPFS node
angelina	AIX 5L V5.3 TML03	KRB5 client	GPFS node
brazos	AIX 5L V5.3 RML03	KRB5 client	
trinity	AIX 5L V5.3 RML03	KRB5 client	
madrid	AIX 5L V5.3 RML03	NIM server	
istanbul	AIX 5L V5.2 RML04	AFS sever	Source for the AFS migration scenario
milan	AIX 5L V5.2 RML04	DFS server	Source for the DFS migration scenario

See Appendix A, “Test environment” on page 309 for a complete diagram of the test systems.

6.2 Infrastructure setup flow

Figure 6-1 on page 121 shows the LDAP and Kerberos V5 server and client configuration flow. Each step must be followed in the order shown in the diagram. The server or servers must be configured prior to the configuration of the client

systems. The test environment created for this book involved both LDAP and Kerberos servers running on the same system; however, this is not mandatory. If LDAP replicas are to be created, perform replica creation immediately after the LDAP server has been configured and before Kerberos configuration is undertaken.

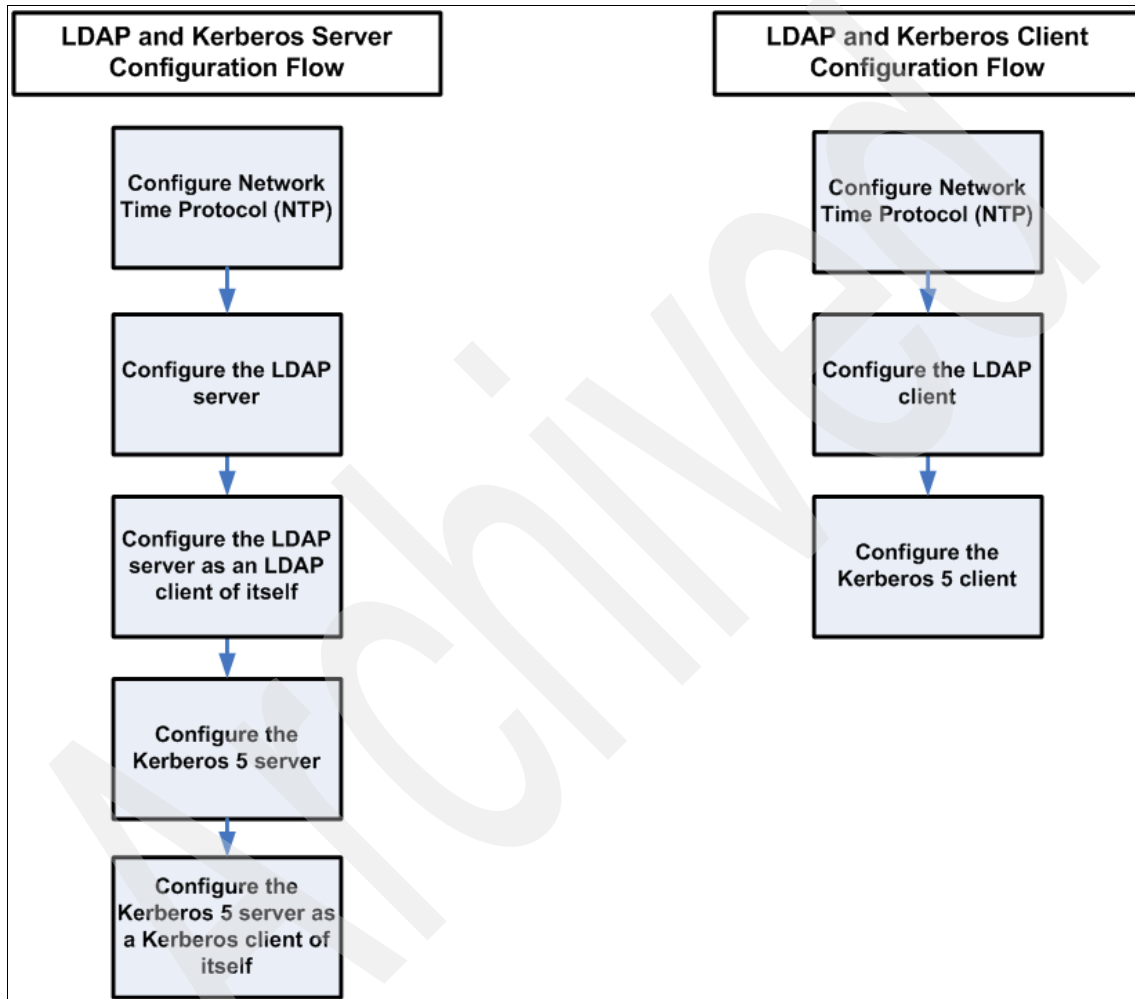


Figure 6-1 Configuration flow for LDAP and Kerberos V5 environments

6.3 Network Time Protocol (NTP) configuration

When using AIX 5L Network Authentication Services (NAS), client and server clocks must be synchronized. This can be accomplished manually or by using locally written scripts. However, a more efficient method involves the use of a Network Time Protocol (NTP) service on AIX 5L. For details about setting up NTP on an AIX 5L system, see Appendix C, “Configuring Network Time Service” on page 329.

6.4 IBM Tivoli Directory Server V5.2

This section describes the steps required to configure an AIX 5L V5.3 system in preparation for deployment of Kerberos V5 with the IBM Tivoli Directory Server LDAP back end.

Important: At the time of writing, a requirement exists for the LDAP server to operate in 64-bit AIX kernel mode. This is required for the db2_08_01.ldap fileset, which provides the LDAP IBM DB2® back end. This might change in future releases of the fileset.

Figure 6-2 on page 123 shows the test environment.

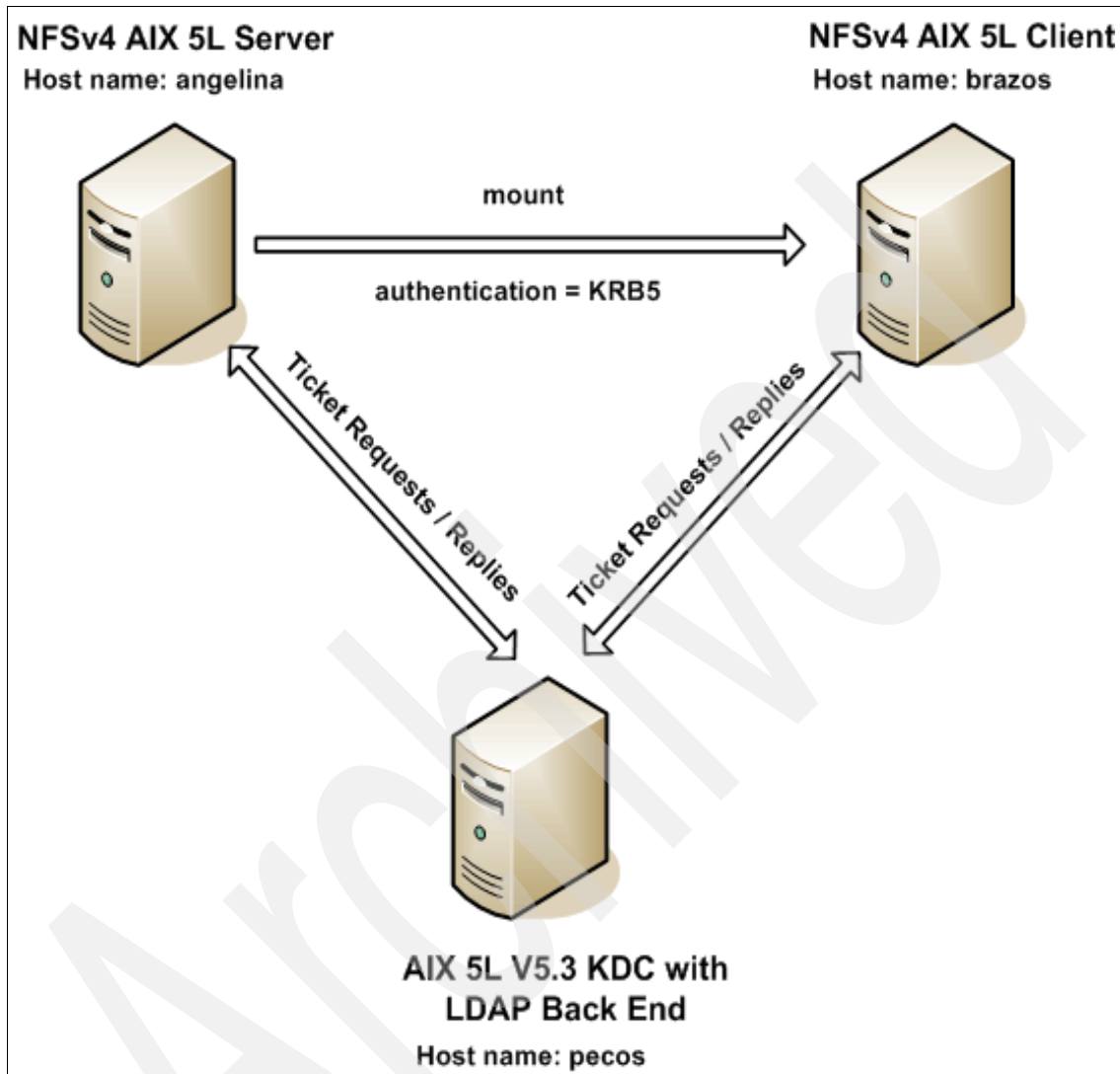


Figure 6-2 Test environment with a key distribution center (KDC) and LDAP back end

Before you install IBM Tivoli Directory Server V5.2, you must prepare the system for the installation.

The following prerequisites are necessary for IBM Tivoli Directory Server:

- ▶ The system must be running in 64-bit AIX kernel mode.
- ▶ Asynchronous I/O must be switched on.

- ▶ At least 500 MB disk space is needed for the Tivoli Directory Server installation directory.
- ▶ You need 200 MB disk space to mount /home/ldapdb2, the ldapdb2 user's home directory.
- ▶ A group called *dbsysadm* must be created, and the *root* user must be added to this group.
- ▶ A user called *ldapdb2* must be created.

6.4.1 Preparing the system for IBM Tivoli Directory Server installation

Verify the following details before beginning the installation of IBM Tivoli Directory Server in order to ensure that the system is correctly configured:

1. Verify that the system is capable of running in 64-bit AIX kernel mode, as shown in Example 6-2.

Example 6-1 Testing for support of 64-bit kernel mode

```
# bootinfo -y
64
#
```

If the result is 32, the system is a 32-bit machine and cannot be used as a Tivoli Directory Server server. If the result is as shown in Example 6-1, continue with the rest of the steps.

2. Check the mode in which the system is currently running, as shown in Example 6-3.

Example 6-2 Testing for the current kernel mode

```
# bootinfo -K
32
#
```

From this output, you can see that the system is currently running in 32-bit mode. Before continuing the installation, it must be configured to run in 64-bit mode. Skip to step 4 if the previous command reports that the system is already in 64-bit mode.

3. Change the system to run in 64-bit kernel mode, as shown in Example 6-3.

Example 6-3 Changing the system to run in 64-bit kernel mode

```
# cd /
#
# ln -sf /usr/lib/boot/unix_64 /unix
#
```

```
# ls -al /unix
lrwxrwxrwx 1 root system 21 Aug 04 15:30 /unix -> /usr/lib/boot/unix_64
#
# bosboot -ad /dev/ipldevice

bosboot: Boot image is 22469 512 byte blocks.
#
```

Normally, the next step involves rebooting the system. However, to save time we reboot after step 4 has been completed.

4. Check if asynchronous I/O (AIO) is enabled, as shown in Example 6-4.

Example 6-4 testing the status of the AIO option

```
# lsattr -El aio0
autoconfig defined STATE to be configured at system restart True
fastpath enable State of fast path True
kprocprio 39 Server PRIORITY True
maxreqs 4096 Maximum number of REQUESTS True
maxservers 10 MAXIMUM number of servers per cpu True
minservers 1 MINIMUM number of servers True
```

Example 6-4 shows that the autoconfig flag is in a defined state. Therefore, AIO is not available. Enabling it across reboots requires setting the autoconfig flag. You can use either the **smitty chgaio** or **chdev** command to make these changes. Example 6-5 shows the **chdev** command syntax. It is followed by the **lsattr** command to confirm the change.

Example 6-5 Using the chdev command to change aio to available

```
# chdev -l aio0 -P -a autoconfig=available
aio0 changed
#
# lsattr -El aio0
autoconfig available STATE to be configured at system restart True
fastpath enable State of fast path True
kprocprio 39 Server PRIORITY True
maxreqs 4096 Maximum number of REQUESTS True
maxservers 10 MAXIMUM number of servers per cpu True
minservers 1 MINIMUM number of servers True
```

5. The system is now rebooted to allow the 64-bit kernel mode to take effect.
6. After the system has rebooted, confirm that it is now running a 64-bit kernel and aio is in an available state (steps 2 and 4).
7. Next, create a 500 MB file system so that the DB2 binaries can be installed. Create the new file system to mount on to /usr/opt/db2_08_01. Change the owner and group to bin.

8. Create a second file system, 200 MB in size, to mount to /home/ldapdb2.

Note: For these tests, a separate volume group was used for the ldapdb2 user's home directory and Tivoli Directory Server installation. This step is not necessary, but the root volume group will be very large if everything is installed in it. This might have an affect on the backup strategy, because many sites normally do not back up the root volume group.

9. Create a group called dbsysadm and add the root user to this group.
10. Create a user called ldapdb2. This user must have the following characteristics:
- Primary group is dbsysadm.
 - Password REGISTRY is files.
 - HOME directory is /home/ldapdb2.
 - Change the owner and group of /home/ldapdb2 to ldapdb2:dbsysadm (use the **chown** command).

Important: Make sure the user ldapdb2 has a valid password and can log in without any challenges before proceeding further.

6.4.2 Installing IBM Tivoli Directory Server

You can now install Tivoli Directory Server. The Tivoli Directory Server filesets are on the fourth CD of the AIX Base Installation media.

Use the following command to install the required filesets:

```
installp -aXYgd /dev/cd0 ldap.server
```

Example 6-6 shows expected results when the installation has completed.

Example 6-6 Final results of the LDAP filesets install

+-----+ Summaries: +-----+				
Installation Summary -----				
Name	Level	Part	Event	Result
db2_08_01.pext	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.msg.en_US.iso88591	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.jhlp.en_US.iso88591	8.1.1.16	USR	APPLY	SUCCESS

db2_08_01.icut	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.icuc	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.db2.samples	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.client	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.cj	8.1.1.16	USR	APPLY	SUCCESS
ldap.client.rte	5.2.0.0	USR	APPLY	SUCCESS
ldap.client.adt	5.2.0.0	USR	APPLY	SUCCESS
ldap.client.rte	5.2.0.0	ROOT	APPLY	SUCCESS
db2_08_01.sqlproc	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.rep1	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.ldap	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.jdbc	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.db2.rte	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.db2.engn	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.das	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.cs.rte	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.conv	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.conn	8.1.1.16	USR	APPLY	SUCCESS
db2_08_01.cnvucs	8.1.1.16	USR	APPLY	SUCCESS
ldap.server.java	5.2.0.0	USR	APPLY	SUCCESS
ldap.server.rte	5.2.0.0	USR	APPLY	SUCCESS
ldap.server.com	5.2.0.0	USR	APPLY	SUCCESS
ldap.server.cfg	5.2.0.0	USR	APPLY	SUCCESS
ldap.server.com	5.2.0.0	ROOT	APPLY	SUCCESS
ldap.server.cfg	5.2.0.0	ROOT	APPLY	SUCCESS
db2_08_01.essg	8.1.1.16	USR	APPLY	SUCCESS
#				

Note: If SSL capabilities are required, the AIX Certificate and SSL base filesets (gskta for 64-bit kernel, gksa for 32-bit kernel) and the crypto filesets from the AIX 5L V5.3 Expansion Pack will also be required. Both the previous filesets are required. Gksa is used by LDAP client applications such as **ldapsearch**.

6.4.3 Configuring IBM Tivoli Directory Server

With Tivoli Directory Server installation complete, we now perform the initial configuration.

First, verify that **ibmslapd** is not running; otherwise, the command in step 1 will fail.

Perform the following steps:

1. Configure the directory and the database using the **mksecldap** command. Example 6-7 on page 128 shows the syntax used in the test environment and the expected output.

Example 6-7 Creating the LDAP server using the mksecdap command

```
# mksecdap -s -a cn=admin -p its0g00d -S rfc2307aix
Filesystem size changed to 229376
Changing password for "ldapdb2"
ldapdb2's New password:
Enter the new password again:
3004-600 The password entry does not match, please try again.
ldapdb2's New password:
Enter the new password again:

You have chosen the following actions:

Administrator DN 'cn=admin' and password will be set.

Setting administrator DN 'cn=admin' and password.
Set administrator DN 'cn=admin' and password.

IBM Tivoli Directory Server Configuration complete.
Enter password for user ldapdb2:
its0g00d

You have chosen the following actions:

Database 'ldapdb2' will be configured in instance 'ldapdb2'.

Configuring IBM Tivoli Directory Server Database.
Creating instance: 'ldapdb2'.
Created instance: 'ldapdb2'.
Cataloging instance node: 'ldapdb2'.
Cataloged instance node: 'ldapdb2'.
Starting database manager for instance: 'ldapdb2'.
Started database manager for instance: 'ldapdb2'.
Creating database: 'ldapdb2'.
Created database: 'ldapdb2'.
Updating the database: 'ldapdb2'
Updated the database: 'ldapdb2'
Updating the database manager: 'ldapdb2'
Updated the database manager: 'ldapdb2'
Enabling multi-page file allocation: 'ldapdb2'
Enabled multi-page file allocation: 'ldapdb2'
Configuring database: 'ldapdb2'
Configured database: 'ldapdb2'
Adding local loop back to database: 'ldapdb2'.
Added local loop back to database: 'ldapdb2'.
Stopping database manager for instance: 'ldapdb2'.
Stopped database manager for instance: 'ldapdb2'.
Starting database manager for instance: 'ldapdb2'.
Started database manager for instance: 'ldapdb2'.
```

Configured IBM Tivoli Directory Server Database.

IBM Tivoli Directory Server Configuration complete.

You have chosen the following actions:

Suffix 'cn=aixdata' will be added to the configuration file.

Adding suffix: 'cn=aixdata'.

Added suffix: 'cn=aixdata'.

IBM Tivoli Directory Server Configuration complete.

Server starting in configuration only mode.

Server starting.

Plugin of type EXTENDEDOP is successfully loaded from libevent.a.

Plugin of type PREOPERATION is successfully loaded from libDSP.a.

Plugin of type PREOPERATION is successfully loaded from libDigest.a.

Plugin of type EXTENDEDOP is successfully loaded from libevent.a.

Plugin of type EXTENDEDOP is successfully loaded from libtranext.a.

Plugin of type AUDIT is successfully loaded from /lib/libldapaudit.a.

Plugin of type EXTENDEDOP is successfully loaded from libevent.a.

Plugin of type DATABASE is successfully loaded from /lib/libback-config.a.

Plugin of type EXTENDEDOP is successfully loaded from liblog.a.

Non-SSL port initialized to 389.

Stopping the LDAP server.

Server starting.

Plugin of type EXTENDEDOP is successfully loaded from libevent.a.

Plugin of type EXTENDEDOP is successfully loaded from libtranext.a.

Plugin of type EXTENDEDOP is successfully loaded from libldaprepl.a.

Plugin of type PREOPERATION is successfully loaded from libDSP.a.

Plugin of type PREOPERATION is successfully loaded from libDigest.a.

Plugin of type EXTENDEDOP is successfully loaded from libevent.a.

Plugin of type EXTENDEDOP is successfully loaded from libtranext.a.

Plugin of type AUDIT is successfully loaded from /lib/libldapaudit.a.

Plugin of type AUDIT is successfully loaded from

/usr/ccs/lib/libsecdapaudit64.a(shr.o).

Plugin of type EXTENDEDOP is successfully loaded from libevent.a.

Plugin of type EXTENDEDOP is successfully loaded from libtranext.a.

Plugin of type DATABASE is successfully loaded from /lib/libback-rdbm.a.

Plugin of type REPLICATION is successfully loaded from /lib/libldaprepl.a.

Plugin of type EXTENDEDOP is successfully loaded from /lib/libback-rdbm.a.

Plugin of type EXTENDEDOP is successfully loaded from libevent.a.

Plugin of type DATABASE is successfully loaded from /lib/libback-config.a.

Plugin of type EXTENDEDOP is successfully loaded from liblog.a.

Non-SSL port initialized to 389.

Migrating users and groups to LDAP server.

#

2. Although the DB2 database is now configured and running, the database must be set to autostart across future system reboots. There are different ways to manage the startup of DB2. One way uses the fault tolerant monitor. In the following steps, we show a different way, using **inittab** and a simple startup script.
 - a. Create a startup script and store it in `/etc/rc.db2`:


```
#!/bin/sh
/usr/bin/su - ldapdb2 -c /home/ldapdb2/sql1lib/adm/db2start
```
 - b. Make the script executable:


```
chmod 755 /etc/rc.db2
```
 - c. Create the inittab entry:


```
mkitab -i ids0 "db2:2:wait:/etc/rc.db2 >/dev/null 2>&1"
```
3. The next step is to tune the DB2 database. Example 6-8 shows the commands used when completing this step.

Example 6-8 Tuning the DB2 database

```
# su - ldapdb2
$
$ db2 update db cfg for ldapdb2 using DBHEAP 20000
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
$
$ db2 update db cfg for ldapdb2 using SORTHEAP 5000
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
$
$ db2 update db cfg for ldapdb2 using APPLHEAPSZ 10000
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
$
$ exit
```

4. Now, stop the Tivoli Directory Server instance to facilitate the addition of the container object for Kerberos using the **ibmdirectl** command. Example 6-9 shows the expected output.

Example 6-9 Stopping the Tivoli Directory Server instance with the ibmdirectl command

```
# ibmdirectl -D cn=admin -w its0g00d stop
Stop operation succeeded
```

5. The next step involves the addition of an LDAP back end to the KDC. Use the **ldapcfg** command:


```
ldapcfg -q -s "o=IBM,c=US"
```

This removes the need to use an established or files-based Kerberos back-end database. For a detailed description of the differences between the

two options, refer to the IBM Network Authentication Service Version 1.4 documentation provided by the krb5.doc.en_US fileset.

- Restart the Tivoli Directory Server instance in order to add the Kerberos schema to the database. Example 6-10 shows the expected output.

Example 6-10 Starting the Tivoli Directory Server instance with the `ibmdirctl` command

```
# ibmdirctl -D cn=admin -w its0g00d start
Start operation succeeded
```

- Add the KRB5 schema, using the `ldapmodify` command:

```
ldapmodify -h pecos.itsc.austin.ibm.com -D cn=admin -w its0g00d -f\
/usr/krb5/ldif/IBM.KRB.schema.ldif -v -c
```

Note: Messages might be generated during this step informing the user that updates failed because certain database entries already exist. These messages can be safely ignored.

- Create a schema for the KDC realm. For this example, we create the file `/usr/ldap/etc/realm_add_ibm.ldif` using the IBM Network Authentication Service Version 1.4 documentation as a reference. Example 6-11 shows the lines added to the new file.

Example 6-11 Sample LDIF file for the KDC realm

```
# cat /usr/ldap/etc/realm_add_ibm.ldif
# The suffix "ou=Austin, o=IBM, c=US" should be defined before attempting to
# load this data. Or change the suffix to be an already defined object.
# Change all references of YOURHOSTNAME.AUSTIN.IBM.COM to be your realm name
#
# version: 1
```

```
dn: o=IBM, c=US
objectclass: top
objectclass: organization
o: IBM
```

```
dn: krbrealmName-V2=NFSV4REALM.IBM.COM, o=IBM, c=US
objectclass: KrbRealm-V2
objectclass: KrbRealmExt
krbrealmName-V2: NFSV4REALM.IBM.COM
krbprincSubtree: krbrealmName-V2=NFSV4REALM.IBM.COM, o=IBM, c=US
krbDeleteType: 3
```

```
dn: cn=principal, krbrealmName-V2=NFSV4REALM.IBM.COM, o=IBM, c=US
```

```
objectclass: container
cn: principal
```

```
dn: cn=policy, krbrealmName-V2=NFSV4REALM.IBM.COM, o=IBM, c=US
objectclass: container
cn: policy
```

The sample file in Example 6-11 on page 131 is available in “Sample LDAP LDIF file for the KDC realm” on page 354.

9. Next, modify the dn: o=IBM, c=US container by adding the schema created in step 8, as shown in Example 6-12.

Example 6-12 Adding the modified realm schema to LDAP

```
# ldapadd -a -h pecos.itsc.austin.ibm.com -D cn=admin -w its0g00d -f\
/usr/ldap/etc/realm_add_ibm.ldif -v -c
ldap_init(pecos.itsc.austin.ibm.com, 389)
add objectclass:
    BINARY (3 bytes) top
    BINARY (12 bytes) organization
add o:
    BINARY (3 bytes) IBM
adding new entry o=IBM, c=US

add objectclass:
    BINARY (11 bytes) KrbRealm-V2
    BINARY (11 bytes) KrbRealmExt
add krbrealmName-V2:
    BINARY (18 bytes) NFSV4REALM.IBM.COM
add krbprincSubtree:
    BINARY (47 bytes) krbrealmName-V2=NFSV4REALM.IBM.COM, o=IBM, c=US
add krbDeleteType:
    BINARY (1 bytes) 3
adding new entry krbrealmName-V2=NFSV4REALM.IBM.COM, o=IBM, c=US

add objectclass:
    BINARY (9 bytes) container
add cn:
    BINARY (9 bytes) principal
adding new entry cn=principal, krbrealmName-V2=NFSV4REALM.IBM.COM, o=IBM, c=US

add objectclass:
    BINARY (9 bytes) container
add cn:
    BINARY (6 bytes) policy
adding new entry cn=policy, krbrealmName-V2=NFSV4REALM.IBM.COM, o=IBM, c=US
```

10. At this point, you can verify the IBM Directory Server using a simple **ldap** query that displays all available container names in the newly created LDAP directory, as shown in Example 6-13.

Example 6-13 Verification of the LDAP namingcontexts

```
# ldapsearch -b "" -s base "objectclass=*" namingcontexts
```

```
namingcontexts=CN=SCHEMA
namingcontexts=CN=LOCALHOST
namingcontexts=CN=PWDPOLICY
namingcontexts=CN=IBMPOLICIES
namingcontexts=CN=AIXDATA
namingcontexts=O=IBM,C=US
```

6.4.4 Configuring Tivoli Directory Server to be a client of itself

Next, we configure the LDAP security client daemon to run on the same system as the LDAP server. To do this, use the following command:

```
mksecldap -c -h pecos.itsc.austin.ibm.com -a cn=admin -p its0g00d
```

You can use the **ls-secldapclntd** command to verify that the client daemon is configured correctly. Example 6-14 shows the expected output.

Example 6-14 Verifying the LDAP client daemon

```
# /usr/sbin/ls-secldapclntd
ldapservers=pecos.itsc.austin.ibm.com
ldapport=389
ldapversion=3
userbasedn=ou=People,cn=aixdata
groupbasedn=ou=Groups,cn=aixdata
idbasedn=ou=System,cn=aixdata
usercachesize=1000
usercacheused=2
groupcachesize=100
groupcacheused=1
cachetimeout=300
heartbeatT=300
numberofthread=10
connectionsperserver=10
alwaysmaster=no
authtype=UNIX_AUTH
searchmode=ALL
defaultentrylocation=LDAP
ldaptimeout=60
userobjectclass=account,posixaccount,shadowaccount,aixauxaccount
groupobjectclass=posixgroup,aixauxgroup
#
```

With the LDAP configuration complete, next we configure the Kerberos server.

6.5 IBM Network Authentication Services (Kerberos V5) server installation

The IBM NAS Version1.4 server filesets are delivered with the AIX 5L V5.3 Expansion Pack CD. You can install these using the **smit** or the **installp** command:

```
installp -aqXgd . krb5.server modcrypt.base
```

Next, we install the NAS client packages:

```
installp -aqXgYd . krb5.lic krb5.client modcrypt.base
```

6.5.1 Setting up the environment

Before proceeding with the NAS configuration, the **PATH** variable must be updated to include the location of the new Kerberos commands. The Kerberos binaries are in the **/usr/krb5/bin** and **/usr/krb5/sbin** directories; update the **/etc/environment** file to include these locations. Example 6-15 shows the updated **PATH** variable.

Example 6-15 Updated PATH variable in /etc/environment

```
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin:/usr/krb5/bin:/usr/krb5/sbin:/usr/java14/jre/bin:/usr/java14/bin
```

It is imperative that you perform this step, because KRB5 commands such as **kinit** are also installed under the Java14.sdk. Therefore, setting the **PATH** variable as shown in Example 6-15 ensures that the correct commands are called.

In a large environment, a copy of the **/etc/environment** file can be distributed to all systems using a locally written shell script in order to eliminate the need to edit each file manually.

6.5.2 Configuring the NAS server

Use the following required steps to configure the NAS server:

1. Set up the environment.
Update the **PATH** variable in the **/etc/environment** file.
2. Configure the NAS server to use Tivoli Directory Server as its back-end database.

Note: The environment created for this test might not suit all needs. Consult the relevant NFSv4 and Network Authentication Services (NAS) material and, if needed, engage IBM Global Services to seek advice about the best way to implement NAS based on your local requirements.

Configure the Kerberos server with an LDAP back end

Use the `mkkrb5srv` or `config.krb5` command to configure NAS to use Tivoli Directory Server as its database back end.

During the execution of this command, the system asks for a Master Database password and a password for the administrative principal admin. Record and store both the name and chosen password in a secure place because they are essential for management of the NAS environment. Example 6-16 shows the output of the `mkkrb5srv` command on the KDC server.

Example 6-16 Output of the mkkrb5srv command on the KDC server

```
# mkkrb5srv -r NFSV4REALM.IBM.COM -s pecos.itsc.austin.ibm.com -d \
itsc.austin.ibm.com -a admin/admin -l pecos.itsc.austin.ibm.com -u "cn=admin" \
-p its0g00d
```

Fileset	Level	State	Description

Path: /usr/lib/objrepos			
krb5.server.rte	1.4.0.1	COMMITTED	Network Authentication Service Server
Path: /etc/objrepos			
krb5.server.rte	1.4.0.1	COMMITTED	Network Authentication Service Server

The -s option is not supported.
The administration server will be the local host.
Initializing configuration...
Creating /etc/krb5/krb5_cfg_type...
Creating /etc/krb5/krb5.conf...
Creating /var/krb5/krb5kdc/kdc.conf...
Creating database files...
Initializing database 'LDAP' for realm 'NFSV4REALM.IBM.COM'
master key name 'K/M@NFSV4REALM.IBM.COM'
Attempting to bind to one or more LDAP servers. This may take a while...
You are prompted for the database Master Password.
It is important that you DO NOT FORGET this password.
Enter database Master Password:
Re-enter database Master Password to verify:
Attempting to bind to one or more LDAP servers. This may take a while...
WARNING: no policy specified for admin/admin@NFSV4REALM.IBM.COM;
defaulting to no policy. Note that policy may be overridden by
ACL restrictions.

```

Enter password for principal "admin/admin@NFSV4REALM.IBM.COM":
Re-enter password for principal "admin/admin@NFSV4REALM.IBM.COM":
Principal "admin/admin@NFSV4REALM.IBM.COM" created.
Creating keytable...
Attempting to bind to one or more LDAP servers. This may take a while...
Creating /var/krb5/krb5kdc/kadm5.acl...
Starting krb5kdc...
Attempting to bind to one or more LDAP servers. This may take a while...
krb5kdc was started successfully.
Starting kadmind...
Attempting to bind to one or more LDAP servers. This may take a while...
kadmind was started successfully.
The command completed successfully.
Restarting kadmind and krb5kdc
Attempting to bind to one or more LDAP servers. This may take a while...
Attempting to bind to one or more LDAP servers. This may take a while...
#

```

The **mkkrb5srv** command also updates the `/etc/inittab` file with the entries shown in Example 6-17. These permit NAS to start automatically at system boot time.

Example 6-17 /etc/inittab entries for the KDC

```

krb5kdc:2:once:/usr/krb5/sbin/krb5kdc
kadm:2:once:/usr/krb5/sbin/kadmind

```

Before configuring the NAS clients, test the Kerberos setup using the following command sequence (see Example 6-18):

1. Verify that all required processes for the Kerberos server have started:

```
ps -ef | grep krb | grep -v grep
```
2. Verify that the admin principal can log in:

```
kinit admin/admin@nfsv4realm.ibm.com
```
3. Verify that the admin principal can obtain a ticket after it has logged in:

```
klist
```

Example 6-18 Basic verification of the Kerberos server

```

# ps -ef | grep krb | grep -v grep
  root 397562      1   0 11:03:26   -   0:00 /usr/krb5/sbin/krb5kdc
  root 585806      1   0 11:03:26   -   0:00 /usr/krb5/sbin/kadmind
#
# kinit admin/admin@NFSV4REALM.IBM.COM
Password for admin/admin@NFSV4REALM.IBM.COM:
#
# klist

```

```
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: admin/admin@NFSV4REALM.IBM.COM
```

```
Valid starting Expires Service principal
08/19/05 11:07:31 08/20/05 11:07:28
krbtgt/NFSV4REALM.IBM.COM@NFSV4REALM.IBM.COM
#
```

Configure the Kerberos server as a client of itself

You must now create the host principals for the Kerberos server. This provides the authority to add, modify, and delete users from the server. Therefore, the server must be configured as a client of itself. Configuring the server as a client of itself also updates the `/usr/lib/security/methods.cfg` file with the stanzas shown in Example 6-19.

Example 6-19 Authentication grammar added to the `/usr/lib/security/methods.cfg` file

```
LDAP:
    program = /usr/lib/security/LDAP
    program_64 = /usr/lib/security/LDAP64

KRB5:
    program = /usr/lib/security/KRB5

KRB5LDAP:
    options = db=LDAP,auth=KRB5
```

Next, we configure integrated login on the server. Use the `mkkrb5clnt` command, as shown in Example 6-20.

Example 6-20 Configuring the Kerberos server as a client of itself

```
# mkkrb5clnt -c pecos.itsc.austin.ibm.com -s pecos.itsc.austin.ibm.com -r\
NFSV4REALM.IBM.COM -d itsc.austin.ibm.com -i LDAP -A -K -T
Initializing configuration...
Creating /etc/krb5/krb5_cfg_type...
Creating /etc/krb5/krb5.conf...
The command completed successfully.
Password for admin/admin@nfsv4realm.ibm.com:
Configuring fully integrated login
Authenticating as principal admin/admin with existing credentials.
WARNING: no policy specified for
host/pecos.itsc.austin.ibm.com@nfsv4realm.ibm.com;
    defaulting to no policy. Note that policy may be overridden by
    ACL restrictions.
Principal "host/pecos.itsc.austin.ibm.com@nfsv4realm.ibm.com" created.
```

Administration credentials NOT DESTROYED.
Making root a Kerberos administrator
Authenticating as principal admin/admin with existing credentials.
WARNING: no policy specified for
root/pecos.itsc.austin.ibm.com@nfsv4realm.ibm.com;
defaulting to no policy. Note that policy may be overridden by
ACL restrictions.
Enter password for principal
"root/pecos.itsc.austin.ibm.com@nfsv4realm.ibm.com":
Re-enter password for principal
"root/pecos.itsc.austin.ibm.com@nfsv4realm.ibm.com":
Principal "root/pecos.itsc.austin.ibm.com@nfsv4realm.ibm.com" created.

Administration credentials NOT DESTROYED.
Authenticating as principal admin/admin with existing credentials.
Principal "host/pecos.itsc.austin.ibm.com@nfsv4realm.ibm.com" deleted.
Make sure that you have removed this principal from all ACLs before reusing.

Administration credentials NOT DESTROYED.
Authenticating as principal admin/admin with existing credentials.
Principal "root/pecos.itsc.austin.ibm.com@nfsv4realm.ibm.com" deleted.
Make sure that you have removed this principal from all ACLs before reusing.

Administration credentials NOT DESTROYED.
Authenticating as principal admin/admin with existing credentials.
Principal "kadmin/admin@nfsv4realm.ibm.com" modified.

Administration credentials NOT DESTROYED.
Cleaning administrator credentials and exiting.

Verify the previous steps by testing whether it is possible to create and modify users and groups. Example 6-21 shows the full command sequence.

Example 6-21 Testing the Kerberos environment

```
# mkuser -R KRB5LDAP testuser
#
# passwd -R KRB5LDAP testuser
Changing password for "testuser"
testuser's Old password:
testuser's New password:
Enter the new password again:
#
# lsuser -R KRB5LDAP testuser
testuser id=212 pgrp=staff groups=staff home=/home/testuser shell=/usr/bin/ksh
login=true su=true rlogin=true telnet=true daemon=true admin=false sugroups=ALL
admggroups= tpath=nosak ttys=ALL expires=0 auth1=SYSTEM auth2=NONE umask=22
registry=KRB5LDAP SYSTEM=KRB5LDAP OR compat logintimes= loginretries=0
pwdwarntime=0 account_locked=false minage=0 maxage=0 maxexpired=-1 minalpha=0
```

```

minother=0 mindiff=0 maxrepeats=8 minlen=0 histexpire=0 histsize=0 pwdchecks=
dictionlist= fsize=2097151 cpu=-1 data=262144 stack=65536 core=2097151
rss=65536 nofiles=2000 time_last_login=0 time_last_unsuccessful_login=0
unsuccessful_login_count=0 roles= krb5_principal=testuser@NFSV4REALM.IBM.COM
krb5_principal_name=testuser@NFSV4REALM.IBM.COM krb5_realm=NFSV4REALM.IBM.COM
maxage=0 expires=0 krb5_last_pwd_change=1124755477 admchk=false
krb5_attributes=requires_preauth
krb5_mod_name=root/pecos.itsc.austin.ibm.com@NFSV4REALM.IBM.COM
krb5_mod_date=1124755477 krb5_kvno=3 krb5_mkvno=0
krb5_max_renewable_life=604800 time_last_login=0 time_last_unsuccessful_login=0
unsuccessful_login_count=0 krb5_names=testuser:pecos.itsc.austin.ibm.com
#
# mkgroup -R KRB5LDAP testgp
#
# lsgroup -R KRB5LDAP testgp
testgp id=204 admin=false users= registry=KRB5LDAP
#
# chuser -R KRB5LDAP pgrp=testgp testuser
#
# lsuser -R KRB5LDAP testuser
testuser id=212 pgrp=testgp groups=testgp,staff home=/home/testuser
shell=/usr/bin/ksh login=true su=true rlogin=true telnet=true daemon=true
admin=false sugroups=ALL admgroups= tpath=nosak ttys=ALL expires=0 auth1=SYSTEM
auth2=NONE umask=22 registry=KRB5LDAP SYSTEM=KRB5LDAP OR compat logintimes=
loginretries=0 pwdwarntime=0 account_locked=false minage=0 maxage=0
maxexpired=-1 minalpha=0 minother=0 mindiff=0 maxrepeats=8 minlen=0
histexpire=0 histsize=0 pwdchecks= dictionlist= fsize=2097151 cpu=-1
data=262144 stack=65536 core=2097151 rss=65536 nofiles=2000 time_last_login=0
time_last_unsuccessful_login=0 unsuccessful_login_count=0 roles=
krb5_principal=testuser@NFSV4REALM.IBM.COM
krb5_principal_name=testuser@NFSV4REALM.IBM.COM krb5_realm=NFSV4REALM.IBM.COM
maxage=0 expires=0 krb5_last_pwd_change=1124755477 admchk=false
krb5_attributes=requires_preauth
krb5_mod_name=root/pecos.itsc.austin.ibm.com@NFSV4REALM.IBM.COM
krb5_mod_date=1124755477 krb5_kvno=3 krb5_mkvno=0
krb5_max_renewable_life=604800 time_last_login=0 time_last_unsuccessful_login=0
unsuccessful_login_count=0 krb5_names=testuser:pecos.itsc.austin.ibm.com
#
# rmuser -R KRB5LDAP testuser
#
# lsuser -R KRB5LDAP testuser | grep pgrp
3004-687 User "testuser" does not exist.
#
# rmgroup -R KRB5LDAP testgp
#
# lsgroup -R KRB5LDAP testgp
3004-686 Group "testgp" does not exist.
#

```

The previous examples verified the following operations:

1. Creating a principal named *testuser*
2. Setting a password for the same user
3. Creating a group called *testgp*
4. Modifying *testuser* to make *testgp* the user's primary group
5. Deleting the user
6. Deleting the group

6.6 IBM Tivoli Directory Server client configuration

All clients must be able to communicate with the LDAP server. The syntax for the **mksecldap** command required to accomplish this task is as follows:

```
mksecldap -c -h <fully_qualified_name_of_LDAP_server> -a <admin_cn> -p\ <password>
```

Example 6-22 shows the use of the **mksecldap** command to create the client, followed by the **ls-secldapclntd** command to verify the configuration.

Example 6-22 Configuring the LDAP client

```
# mksecldap -c -h pecos.itsc.austin.ibm.com -a cn=admin -p its0g00d
#
# ls-secldapclntd
ldapservers=pecos.itsc.austin.ibm.com
ldapport=389
ldapversion=3
userbasedn=ou=People,cn=aixdata
groupbasedn=ou=Groups,cn=aixdata
idbasedn=cn=aixid,ou=System,cn=aixdata
usercachesize=1000
usercacheused=2
groupcachesize=100
groupcacheused=1
cachetimeout=300
heartbeatT=300
numberofthread=10
connectionsperserver=10
alwaysmaster=no
authtype=UNIX_AUTH
searchmode=ALL
defaultentrylocation=LDAP
ldaptimeout=60
userobjectclass=account,posixaccount,shadowaccount,aixauxaccount,ibm-securityId
entities
groupobjectclass=posixgroup,aixauxgroup
#
```

6.7 IBM Network Authentication Services client install and configuration

Now, we configure the IBM NAS client environment. The following tasks must be performed:

- ▶ Set up the environment.
Update the PATH variable in the /etc/environment file. If you have not done this yet, refer to 6.5.1, “Setting up the environment” on page 134.
- ▶ Configure the NAS client or clients. Two methods are available:
 - Integrated login (single sign-on)
 - Standard login

Clients can only be configured to use *one* of these methods.

6.7.1 Integrated login (single sign-on)

Integrated login can be configured to attempt user authentication through one or more methods, such as the /etc/passwd file, DCE, AFS, or Kerberos, at login time. Enabling integrated login based on the distributed Kerberos client/server architecture, which uses a centrally stored user database, eases the administrative burden by removing the requirement that each user be managed at a per system level (that is, the /etc/passwd file). Instead, login attempts will first attempt to authenticate the user against the Kerberos subsystem. Use of this method provides the added benefit that users no longer are required to remember multiple user IDs and passwords; their Kerberos ID can be used when logging in to any machine located in the realm.

Example 6-23 uses the **mkkrb5clnt** command to configure a client to use integrated login.

Example 6-23 Configuring the NAS client to use integrated login

```
# mkkrb5clnt -c pecos.itsc.austin.ibm.com -s pecos.itsc.austin.ibm.com -r\
NFSV4REALM.IBM.COM -d itsc.austin.ibm.com -i LDAP -A -K -T
Initializing configuration...
Creating /etc/krb5/krb5_cfg_type...
Creating /etc/krb5/krb5.conf...
The command completed successfully.
Password for admin/admin@NFSV4REALM.IBM.COM:
Configuring fully integrated login
Authenticating as principal admin/admin with existing credentials.
WARNING: no policy specified for
host/brazos.itsc.austin.ibm.com@NFSV4REALM.IBM.COM;
defaulting to no policy. Note that policy may be overridden by
```

ACL restrictions.
Principal "host/brazos.itsc.austin.ibm.com@NFSV4REALM.IBM.COM" created.

Administration credentials NOT DESTROYED.
Making root a Kerberos administrator
Authenticating as principal admin/admin with existing credentials.
WARNING: no policy specified for
root/brazos.itsc.austin.ibm.com@NFSV4REALM.IBM.COM;
defaulting to no policy. Note that policy may be overridden by
ACL restrictions.
Enter password for principal
"root/brazos.itsc.austin.ibm.com@NFSV4REALM.IBM.COM":
Re-enter password for principal
"root/brazos.itsc.austin.ibm.com@NFSV4REALM.IBM.COM":
Principal "root/brazos.itsc.austin.ibm.com@NFSV4REALM.IBM.COM" created.

Administration credentials NOT DESTROYED.
Configuring Kerberos as the default authentication scheme
Cleaning administrator credentials and exiting.
#

The final step is to change the authentication grammar on the client. This must be done for all users, except the *root* user. For the test environment, the Password Registry entry for the root user was changed to files and the authentication grammar to compat. Example 6-24 shows how this was accomplished.

Example 6-24 Setting the root user's authentication grammar

```
# chuser registry=files root
#
# chuser SYSTEM="compat" root
#
# grep -p root /etc/security/user
root:
    admin = true
    SYSTEM = "compat"
    registry = files
    loginretries = 0
    account_locked = false
    admgroups =
```

For all other users, set the registry entry to KRB5LDAP and SYSTEM to "KRB5LDAP OR compat", as shown in Example 6-25 on page 143. It might also be useful to perform the previous operation on other AIX 5L *system* users, such as bin, lp, or etc.

Example 6-25 Setting the authentication grammar for other users

```
# chsec -f /etc/security/user -s default -a registry=KRB5LDAP
#
# chsec -f /etc/security/user -s default -a "SYSTEM=\"KRB5LDAP OR compat\""
#
# grep -p default /etc/security/user
default:
    admin = false
    login = true
    su = true
    daemon = true
    rlogin = true
    sugroups = ALL
    admggroups =
    ttys = ALL
    auth1 = SYSTEM
    auth2 = NONE
    tpath = nosak
    umask = 022
    expires = 0
    SYSTEM = "KRB5LDAP OR compat"
    logintimes =
    pldwarntime = 0
    account_locked = false
    loginretries = 0
    histexpire = 0
    histsize = 0
    minage = 0
    maxage = 0
    maxexpired = -1
    minalpha = 0
    minother = 0
    minlen = 0
    mindiff = 0
    maxrepeats = 8
    dictionlist =
    pwdchecks =
    registry = KRB5LDAP
#
```

Our client is now configured to use integrated login.

6.7.2 Standard login

This method might be useful in cases where not all users participate in a Kerberized environment. The initial login uses the standard AIX authentication method, that is, the `/etc/passwd` and `/etc/security/passwd` files. When a user

requires access to a file system or other resource requiring Kerberos authentication, they must obtain a *ticket* using the **kinit** command. This method introduces a major disadvantage, because it means that each user must have a local entry (and therefore, potentially, a different password) on each system.

Example 6-26 uses the **mkkrb5clnt** command to configure the client for NAS, but continues to use standard AIX login methods.

Example 6-26 Configuring the client for NAS but using standard AIX login

```
# mkkrb5clnt -c pecos.itsc.austin.ibm.com -s pecos.itsc.austin.ibm.com -r\  
nfsv4realm.ibm.com -d itsc.austin.ibm.com  
Initializing configuration...  
Creating /etc/krb5/krb5_cfg_type...  
Creating /etc/krb5/krb5.conf...  
The command completed successfully.
```

After the client is configured, the **kinit** and **klist** commands are used to verify that it can authenticate to the server, as shown in Example 6-27.

Example 6-27 Verifying kerberos client operation

```
# kinit admin/admin@NFSV4REALM.IBM.COM  
Password for admin/admin@nfsv4realm.ibm.com:  
# klist  
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0  
Default principal: admin/admin@nfsv4realm.ibm.com  
  
Valid starting Expires Service principal  
08/16/05 09:22:11 08/17/05 09:22:07  
krbtgt/nfsv4realm.ibm.com@nfsv4realm.ibm.com
```

6.7.3 Adding NAS users

This section discusses the methods available for adding users to an NAS environment. Several options are available. For example, the **mkuser** command is a simplified front end that enables administrators to create accounts located in the `/etc/passwd` file, Kerberos, or another authentication database. The database used for the account is determined by the value of the **-R** switch; if this switch is unspecified, the default authentication method is used.

Example 6-28 on page 145 uses the **mkuser** and **passwd** commands with the **-R KRB5LDAP** option to create NAS users.

Example 6-28 Creating NAS users

```
# kinit admin/admin
Password for admin/admin@nfsv4realm.ibm.com:
#
# mkuser -R KRB5LDAP <user_name>
#
# passwd -R KRB5LDAP <user_name>
```

It is also possible to create Kerberos user principals matching existing UNIX user names. The principal name is mapped to the user name by NFS to determine the UNIX credential associated with the principal. You can add principals the KDC server using the command sequence:

```
kadmin.local -> add_principal (or addprinc) <principal_name>.
```

Note: The **kadmin.local** command can only be run on the master KDC, while **kadmin** can be run on any machine that is part of the Kerberos realm.

In Example 6-29, the principal named *jen* is added to the KDC database from the command line. Because the **kadmin.local** command suite is privileged, this operation requires prior authentication to Kerberos using an administrative principal, for example, *admin/admin*.

Example 6-29 Adding the principal named *jen* to the KDC

```
# kadmin.local
kadmin.local: addprinc -e des-cbc-crc:normal jen
WARNING: no policy specified for jen@nfsv4realm.ibm.com;
        defaulting to no policy. Note that policy may be overridden by
        ACL restrictions.
Enter password for principal "jen@nfsv4realm.ibm.com":
Re-enter password for principal "jen@nfsv4realm.ibm.com":
Principal "jen@nfsv4realm.ibm.com" created.
kadmin.local:
```

The newly created principal must now be verified. Principals can be listed on KDC using the following command sequence:

```
kadmin.local -> list_principals
```

The **kadmin.local** interface also provides the options **get_principal** and **getprinc** to generate account details for a specific user. Example 6-30 on page 146 shows the results of a **getprinc** command for the principal *jen*.

Example 6-30 Listing details for principal jen

```
kadmin.local: getprinc jen
Principal: jen@nfsv4realm.ibm.com
Expiration date: [never]
Last password change: Tue Aug 16 16:35:09 CDT 2005
Password expiration date: [none]
Maximum ticket life: 1 day 00:00:00
Maximum renewable life: 7 days 00:00:00
Last modified: Tue Aug 16 16:35:09 CDT 2005 (admin/admin@nfsv4realm.ibm.com)
Last successful authentication: [never]
Last failed authentication: [never]
Failed password attempts: 0
Number of keys: 1
Key: vno 1, DES cbc mode with CRC-32,
no salt

Attributes:
  REQUIRES_PRE_AUTH
Policy: [none]
kadmin.local:
```

Finally, verify the newly created Kerberos principal using the **klist** command.

Example 6-31 Verify the principal jen with using the klist command

```
# kinit jen@nfsv4realm.ibm.com
Password for jen@nfsv4realm.ibm.com:
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: jen@nfsv4realm.ibm.com

Valid starting    Expires          Service principal
08/16/05 16:52:05 08/17/05 16:52:03  krbtgt/nfsv4realm.ibm.com@nfsv4realm.ibm.com
```

Use of the previous procedure in environments involving a large number of users can be cumbersome and time-consuming. The script shown in Example 6-32 on page 147 might assist this. It assumes that a file containing a list of users to be added to the KDC has already been created; this example uses the file `users.out` as the source for user names. You can use the **kadmin.local** command if the script is executed on the NAS server; **kadmin** is required when a client machine is used.

Example 6-32 Creating user principals using a shell script

```
#!/bin/ksh
NPASSWD="new01new"
for i in `cat users.out`
do
/usr/krb5/sbin/kadmin.local <<EOF
add_principal -e des-cbc-crc:normal -pw ${NPASSWD} $i
EOF
```

6.7.4 Migrating existing users into NAS

Many sites conducting a migration to an NAS environment require the ability to migrate existing UNIX user names, user IDs, and any additional information resident in the `/etc/passwd` file. Migration of passwords is not possible. However, a temporary password can be set for migrated user account, and a flag set on the account to force the user to change this password during their initial login session.

The script in Example 6-33 uses the `mkuser`, `kadmin.local`, and `chuser` commands to automate this task. It takes a copy of a host system's `/etc/security/passwd` file as input, and uses an inline `awk` statement to create an output file, `users_passwd.out`. The script also assumes that a copy of the host system's `/etc/security/passwd` file has been amended to remove standard AIX users such as `root`, `lp`, and `bin`. The example script is by no means complete or fault tolerant, but provides a starting point for sites requiring this capability.

The following steps were performed:

1. A copy of the host system's `/etc/security/passwd` file was transferred to one of the test realm's NAS clients.
2. The script in Example 6-33 was executed.
3. The `kadmin.local -> listprincs` commands were used to verify that the users were correctly migrated.
4. Several of the migrated accounts were tested using `telnet` and other commands to ensure that it was possible to log in to them successfully.

The format of the `users_passwd.out` file is as follows:

```
userid:encrypted_passwd
```

Example 6-33 Migrating or adding users to the KDC and LDAP

```
#!/bin/ksh
#
# Script to migrate user information
# The script does the following:
```

```

#
# i. adds the user
# ii. displays user information in LDAP
# iii. sets the user's temporary password
# iv. expires the user's password, forcing a change at first log on
#
export AUTH=KRB5LDAP
export PASSWD=tempONEtemp

# if the admusr user does not exist, create it.
lsuser -R $AUTH admusr || /usr/bin/mkuser -R $AUTH -a account_locked=true
admin=true admusr

# extract the username, UID and gecos information from the input file
cat /mnt/user_data.out | awk -F":" '

# add user routine with UID & gecos info
adduser=sprintf("/usr/bin/mkuser -R $AUTH -a id=%s gecos=\"%s\" %s", $2, $3,
$1)

# display user information from LDAP routine
lsuser=sprintf("/usr/sbin/lsuser -R $AUTH %s", $1)

# set the the temporary password routine
passwd=sprintf("/usr/krb5/sbin/kadmin.local -q \"change_password -pw $PASSWD
%s\"", $1)

# expire the temporary password routine
chuser=sprintf("/usr/bin/chuser -R $AUTH krb5_attributes=+needchange %s", $1)

# run the routines
system(adduser)
system(lsuser)
system(passwd)
system(chuser)
'

```

Note: Users will be asked to change their password the first time they log in after the migration. No other method exists to migrate the users' passwords without writing custom programs to manipulate the KDC directly.

Example 6-34 on page 149 shows the output when a user logs in for the first time after the migration. System integrity and security are preserved, because users must use their existing password to gain access to the system. They are then required to change this password. It also means that the systems administrator is not required to painstakingly assign a temporary password to each migrated user account.

Example 6-34 First time login for a migrated user

```
AIX Version 5
(C) Copyrights by IBM and by others 1982, 2005.
login: max
max's Password:
[compat]: 3004-610 You are required to change your password.
        Please choose a new one.

max's New password:
Enter the new password again:
```

6.7.5 Installation details

After the Kerberos installation and configuration is complete, the files shown in Table 6-2 are installed in the /etc/krb5 directory.

Table 6-2 Kerberos config files

File	Description
krb5.conf	The krb5.conf file contains general information for clients and servers. It must reside on each system containing the administration server, a KDC, or client. If two or more Network Authentication Service servers or clients reside on the same system, they must share the same krb5.conf file.
krb5_cfg_type	The krb5_cfg_type file determines the configuration type of the machine (master, slave, or client). This file must reside on the system that contains the administration server.

Example 6-35 and Example 6-36 on page 150 show the contents of the 5/krb5.conf and /etc/krb5/krb5_cfg_type files.

Example 6-35 Contents of the /etc/krb5.conf file

```
[libdefaults]
    default_realm = NFSV4REALM.IBM.COM
    default_keytab_name = FILE:/etc/krb5/krb5.keytab
    default_tkt_enctypes = des3-cbc-sha1 arcfour-hmac aes256-cts
des-cbc-md5 des-cbc-crc
    default_tgs_enctypes = des3-cbc-sha1 arcfour-hmac aes256-cts
des-cbc-md5 des-cbc-crc
    use_ldap_lookup = 1
    ldap_server = pecos.itsc.austin.ibm.com

[realms]
    NFSV4REALM.IBM.COM = {
        kdc = pecos.itsc.austin.ibm.com:88
```

```

        admin_server = pecos.itsc.austin.ibm.com:749
        default_domain = itsc.austin.ibm.com
    }

[domain_realm]
    .itsc.austin.ibm.com = NFSV4REALM.IBM.COM
    pecos.itsc.austin.ibm.com = NFSV4REALM.IBM.COM

[logging]
    kdc = FILE:/var/krb5/log/krb5kdc.log
    admin_server = FILE:/var/krb5/log/kadmin.log
    default = FILE:/var/krb5/log/krb5lib.log

```

Note: The `default_tkt_enctypes` and `default_tgs_enctypes` can be reduced to show only `des-cbc-crc` and `des-cbc-md5`, but this might only be correct with NFSv4.

Example 6-36 Contents of the `/etc.krb5_cfg_type` file

```

# cat /etc/krb5_cfg_type
master

```

6.8 Installing GPFS

GPFS is not mandatory for a successful deployment of NFSv4. However, we discuss it in this chapter in order to assist sites that need to take advantage of the resiliency features it provides. GPFS can be used to great advantage if a highly resilient NFSv4 environment is required; it is especially helpful when used in conjunction with the read/write replication facility provided by NFSv4.

The installation procedures are generalized for all levels of GPFS. Ensure that the correct numeric value is provided for the modification (m) and fix (f) levels, where applicable. The modification and fix level are dependent on the level of PTF support.

Three GPFS nodes are used for this example:

- ▶ `sabine.itsc.austin.ibm.com`
- ▶ `frio.itsc.austin.ibm.com`
- ▶ `angelina.itsc.austin.ibm.com`

One installation system is also present:

- ▶ `madrid.itsc.austin.ibm.com`

Follow the steps in this section to install the GPFS software using the `installp` command.

Important: The examples are based on a new installation. If upgrading from an earlier version of GPFS, consult the necessary documentation to assist with reconciling the differences with the new version.

6.8.1 Preparing the GPFS nodes for installation

These examples use GPFS V2.3. Before GPFS can be installed on each node, the `/.rhosts` file must contain an entry for *madrid* because it is the source of the installation media. The following line was added to the `/.rhosts` file on *sabine*, *frio*, and *angelina*:

```
madrid.itsc.austin.ibm.com
```

The file permissions on the `/.rhosts` file must be 644, as shown in Example 6-37.

Example 6-37 Setting the permissions on the `/.rhosts` file

```
# chmod 644 /.rhosts
#
# ls -la /.rhosts
-rw-r--r--  1 root      system      91 Aug 09 18:14 /.rhost
```

To ease the installation process, a file named `/tmp/gpfs.allnodes` was created on *madrid* containing the full names of all GPFS nodes. Example 6-38 shows the contents of the `/tmp/gpfs.allnodes` file. We use this file when installing GPFS on *frio* and *sabine*.

Example 6-38 Contents of the `/tmp/gpfs.allnodes` file

```
frio.itsc.austin.ibm.com
sabine.itsc.austin.ibm.com
```

It is now possible to proceed with the GPFS installation.

6.8.2 Creating the GPFS directory

Perform the following steps to create the GPFS directory:

1. On *madrid*, create a temporary subdirectory where the GPFS installation images can be extracted:

```
# mkdir /tmp/gpfs1pp
```

2. Copy the installation images from the CD-ROM to the new directory using the **bffcreate** command. The following example shows this process on madrid:

```
# bffcreate -qvX -t /tmp/gpfs1pp -d /dev/cd0 all
```

3. Step 2 places the following images in /tmp/gpfs1pp:

- gpfs.base
- gpfs.docs
- gpfs.msg.en_US

Tip: The gpfs.docs image must be installed in order to use the GPFS man pages. The GPFS manual pages are in the /usr/share/man/ directory. The gpfs.docs image is not necessary on all nodes if the man pages are not required or file system space on the node is limited.

6.8.3 Creating the GPFS installation table of contents file

The next step is to create the installation table of contents (.toc), as shown in Example 6-39.

Example 6-39 Create the .toc file

```
# cd /tmp/gpfs1pp
#
# inutoc .
#
# # ls -la .toc
-rw-r--r-- 1 root    system      3153 Aug 09 18:49 .toc
#
```

6.8.4 Installing GPFS through the network

If you use a network installation, first ensure that the directory where the GPFS images are (for example, /gpfs_tools/gpfs.allnodes) is NFS exported to all nodes designated for the GPFS cluster.

Ensure that an acceptable directory or mount point is available on each target node, such as /tmp/gpfs1pp. If it does not exist, create it:

```
# WCOLL=/gpfs_tools/gpfs.allnodes dsh "mkdir /tmp/gpfs1pp"
```

If installing on a shared file system, place the GPFS images on each node in the network by running:

```
# WCOLL=/gpfs_tools/gpfs.allnodes dsh "mount madrid:/tmp/gpfs1pp /tmp/gpfs1pp"
```

Otherwise, run:

```
# WCOLL=/gpfs_tools/gpfs.allnodes dsh "rcp madrid:/tmp/gpfs1pp/gpfs*  
/tmp/gpfs1pp"  
#  
# WCOLL=/gpfs_tols/gpfs.allnodes dsh "rcp madrid:/tmp/gpfs1pp/.toc  
/tmp/gpfs1pp"
```

We can now install GPFS on each node by running:

```
# WCOLL=/gpfs_tools/gpfs.allnodes dsh "installp -agXYd /tmp/gpfs1pp gpfs"
```

6.8.5 Verifying the GPFS installation

Verify the installation after it is complete using the `ls1pp -l gpfs*` command on all nodes. Example 6-40 shows the expected output from a newly installed GPFS node.

Example 6-40 Sample output from the ls1pp command

ls1pp -l gpfs*

Fileset	Level	State	Description

Path: /usr/lib/objrepos			
gpfs.base	2.3.0.0	COMMITTED	GPFS File Manager
gpfs.msg.en_US	2.3.0.0	COMMITTED	GPFS Server Messages - U.S. English
Path: /etc/objrepos			
gpfs.base	2.3.0.0	COMMITTED	GPFS File Manager
Path: /usr/share/lib/objrepos			
gpfs.docs.data	2.3.0.0	COMMITTED	GPFS Server Manpages and Documentation

Important: The use of GPFS to export NFSv4 volumes requires the presence of GPFS V2.3.0.7 or later.

6.9 Configuring GPFS

The GPFS environment must now be configured. We perform the following steps:

1. Set up the environment:
 - a. Update the PATH variable in /etc/environment.
 - b. Update the /.rhosts file on all GPFS nodes.

- c. Create an NFS export for all nodes. The export holds generic configuration information.
- d. Create the all_nodes file.
2. Create the GPFS cluster.
3. Create the GPFS file system.

Note: The environment created for this test scenario might not suit all needs. Consult the relevant GPFS material and, if needed, engage IBM Global Services to seek advice about the best way to implement GPFS in your local environment.

6.9.1 Setting up the environment

Perform the following steps to set up the environment:

1. All GPFS binaries are in /usr/lpp/mmfs/bin, which should be added to the PATH environment variable using either the local .kshrc file or to the /etc/environment file. Example 6-41 shows the results of the latter option.

Example 6-41 Adding the GPFS binaries location to the PATH variable

```
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin:/usr/krb5/bin:/usr/java14/jre/bin:/usr/java14/bin:/usr/lpp/mmfs/bin
```

2. All nodes that are to participate in the GPFS cluster must be added to the .rhosts file on *all* nodes. Example 6-42 shows the contents of the .rhosts file on both GPFS nodes in the test environment.

Example 6-42 Contents of the .rhosts file on all nodes that will be part of GPFS

```
sabine.itsc.austin.ibm.com  
frio.itsc.austin.ibm.com  
angelina.itsc.austin.ibm.com
```

3. To simplify the management of files common to all nodes in the cluster, an NFS export was created on one node and mounted on all other nodes. For this example, we use the /gpfs_tools directory. The choice of NFS protocol is left to the user.
4. It is necessary to create a file containing the names of *all* GPFS nodes and the roles they fulfill in the cluster. For this purpose, a file called gpfs_nodefile was generated in the shared /gpfs_tools directory. Example 6-43 on page 155 shows the contents of the gpfs_nodefile.

```
frio.itsc.austin.ibm.com:quorum
sabine.itsc.austin.ibm.com:quorum
angelina.austin.ibm.com:quorum
```

6.9.2 Creating the GPFS cluster and nodes

We can now create the GPFS cluster using the **mmcrcluster** command, after which each node will be defined. The following steps show the procedure used to create the test GPFS cluster, consisting of systems sabine and frio:

1. Create the cluster, as shown in Example 6-44.

Example 6-44 Creating the GPFS cluster

```
# mmcrcluster -n /gpfs_tools/gpfs.allnodes -p angelina -C NFSv4_GPFS
Fri Aug 12 15:36:08 CDT 2005: 6027-1664 mmcrcluster: Processing node
frio.itsc.austin.ibm.com
Fri Aug 12 15:36:09 CDT 2005: 6027-1664 mmcrcluster: Processing node
sabine.itsc.austin.ibm.com
Fri Aug 12 15:36:09 CDT 2005: 6027-1664 mmcrcluster: Processing node
angelina.itsc.austin.ibm.com
mmcrcluster: Command successfully completed
mmcrcluster: 6027-1371 Propagating the changes to all affected nodes.
This is an asynchronous process.
```

2. Verify the newly created cluster by executing the **mmllsconfig** command on all nodes, as shown in Example 6-45.

Example 6-45 Verifying the created GPFS cluster

```
# mmllsconfig
Configuration data for cluster NFSv4_GPFS.itsc.austin.ibm.com:
-----
clusterName NFSv4_GPFS.itsc.austin.ibm.com
clusterId 649369025867640738
clusterType lc
multinode yes
autoload no
useDiskLease yes
maxFeatureLevelAllowed 809

File systems in cluster NFSv4_GPFS.itsc.austin.ibm.com:
-----
none
```

3. Start GPFS on all nodes in the cluster using the **mmstartup** command, as shown in Example 6-46 on page 156.

Example 6-46 Starting GPFS

```
# mmstartup -a

Thu Aug 11 10:43:24 CDT 2005: 6027-1642 mmstartup: Starting GPFS ...
#
# ps -ef | grep mm
  root  77828 344256   3 15:37:57   -  0:00
/usr/lpp/mmfs/bin/aix64/mmfsd64
  root 131190 323746   0 15:37:59 pts/0  0:00 grep mm
  root 151650 225512   0 10:20:04   -  0:00 /usr/sbin/rsct/bin/rmcd -a
IBM.LPCCommands -r
  root 245842      1   0 15:36:12 pts/0  0:00 /usr/bin/perl -w
/usr/lpp/mmfs/bin/mmgetobjd 6669 0:2:
  root 344256      1   2 15:37:57   -  0:00 /bin/ksh
/usr/lpp/mmfs/bin/runmmfs
```

6.9.3 Creating a GPFS file system

With the GPFS cluster successfully created, next we create a GPFS file system. Perform the following steps:

1. Create a file called `/gpfs_tools/disk.desc`. This file contains a list of disks to be used when creating GPFS file systems. See Example 6-47.

Example 6-47 The `disk.desc` file

```
hdisk1:::1
hdisk2:::1
```

2. Create cluster-wide names for Network Shared Discs (NSDs) used by GPFS, as shown in Example 6-48. Use the `mmcrnsd` command.

Example 6-48 Create the NSDs

```
# mmcrnsd -F /gpfs_tools/disk.desc

mmcrnsd: Processing disk hdisk1
mmcrnsd: Processing disk hdisk2
mmcrnsd: 6027-1371 Propagating the changes to all affected nodes.
This is an asynchronous process.
```

Tip: If the disks were previously used for GPFS file systems, use the `-v no` option with the `mmcrnsd` command.

The `mmcrnsd` command modifies the `/gpfs_tool/disk.desc` file. Example 6-49 on page 157 shows the modified `disk.desc` file.

Example 6-49 Modified disk.desc file

```
# hdisk1:::1
gpfs3nsd:::dataAndMetadata:1
# hdisk2:::1
gpfs4nsd:::dataAndMetadata:1
```

3. Now, create a GPFS file system using the `mmcrfs` command. Example 6-50 shows the output from the GPFS created in the test environment.

Example 6-50 Creating the GPFS

```
# mmcrfs /gpfs1 gpfs1 -F /gpfs_tools/disk.desc -D nfs4 -k all
```

GPFS: 6027-531 The following disks of gpfs1 will be formatted on node
sabine.itsc.austin.ibm.com:
 gpfs3nsd: size 573330948 KB
 gpfs4nsd: size 573330948 KB
GPFS: 6027-540 Formatting file system ...
Creating Inode File
 58 % complete on Thu Aug 11 13:55:55 2005
 100 % complete on Thu Aug 11 13:55:58 2005
Creating Allocation Maps
Clearing Inode Allocation Map
Clearing Block Allocation Map
Flushing Allocation Maps
GPFS: 6027-535 Disks up to size 1.2 TB can be added to this file system.
GPFS: 6027-572 Completed creation of file system /dev/gpfs1.
mmcrfs: 6027-1371 Propagating the changes to all affected nodes.
This is an asynchronous process.

Note: It is important to choose the correct options when creating the GPFS.

The `mmcrfs` command has the following options:

<code>/gpfs</code>	Mount point for the GPFS.
<code>gpfs1</code>	The name of the device to be used.
<code>-F /gpfs_tools/disk.desc</code>	The full path to the disk description file.

- D nfs4** Specifies whether a “deny-write open lock” will block writes, which is expected and required by NFSv4. File systems supporting NFSv4 must have **-D nfs4** set. The option **-D posix** allows NFS writes even in the presence of a deny-write open lock. If we intend to export the file system using NFSv4, we must use **-D nfs4**. For NFSv3 (or if the file system is not NFS exported at all), use **-D posix**.
- k**
- all** Specifies the type of authorization supported by the file system.
- posix** Traditional GPFS ACLs only (NFSv4 ACLs are not allowed). Authorization controls are unchanged from earlier releases. The default is **-k posix**.
- nfs4** Support for NFSv4 ACLs only. Users are not allowed to assign traditional GPFS ACLs to any file system objects (directories and individual files).
- all** Any supported ACL type is permitted. This includes traditional GPFS (**posix**) and NFSv4 ACLs (**nfs4**). The administrator is allowing a mixture of ACL types. For example, fileA might have a posix ACL, while fileB in the same file system might have an NFSv4 ACL, implying different access characteristics for each file depending on the ACL type that is currently assigned.

Note: Neither **nfs4** nor **all** should be specified unless the file system is to be exported to NFSv4 clients. NFSv4 ACLs affect file attributes file attributes (mode) and have access and authorization characteristics that are different from traditional GPFS ACLs.

4. The newly created GPFS is now ready to be mounted and its characteristics examined. Use the **mount** and **df -k** commands to perform these operations. See Example 6-51.

Example 6-51 df -k output after the GPFS file system is mounted on the nodes

```
# mount /gpfs1
#
# df -k
Filesystem      1024-blocks      Free %Used      Iused %Iused Mounted on
/dev/hd4         65536           52644    20%         1559    12% /
/dev/hd2        1114112          71964    94%        24625    58% /usr
/dev/hd9var       65536           58576    11%          347     3% /var
/dev/hd3         131072           79564    40%          32     1% /tmp
```

/dev/hd1	65536	65180	1%	5	1% /home
/proc	-	-	-	-	- /proc
/dev/hd10opt	65536	37724	43%	655	8% /opt
/dev/fs1v00	1048576	1048048	1%	27	1% /work
/dev/gpfs1	1146661888	1146238208	1%	39	1% /gpfs1

5. Mount the GPFS on all nodes using the commands in step 4.

Figure 6-3 shows the layout of test environment created to demonstrate NFSv4 over GPFS.

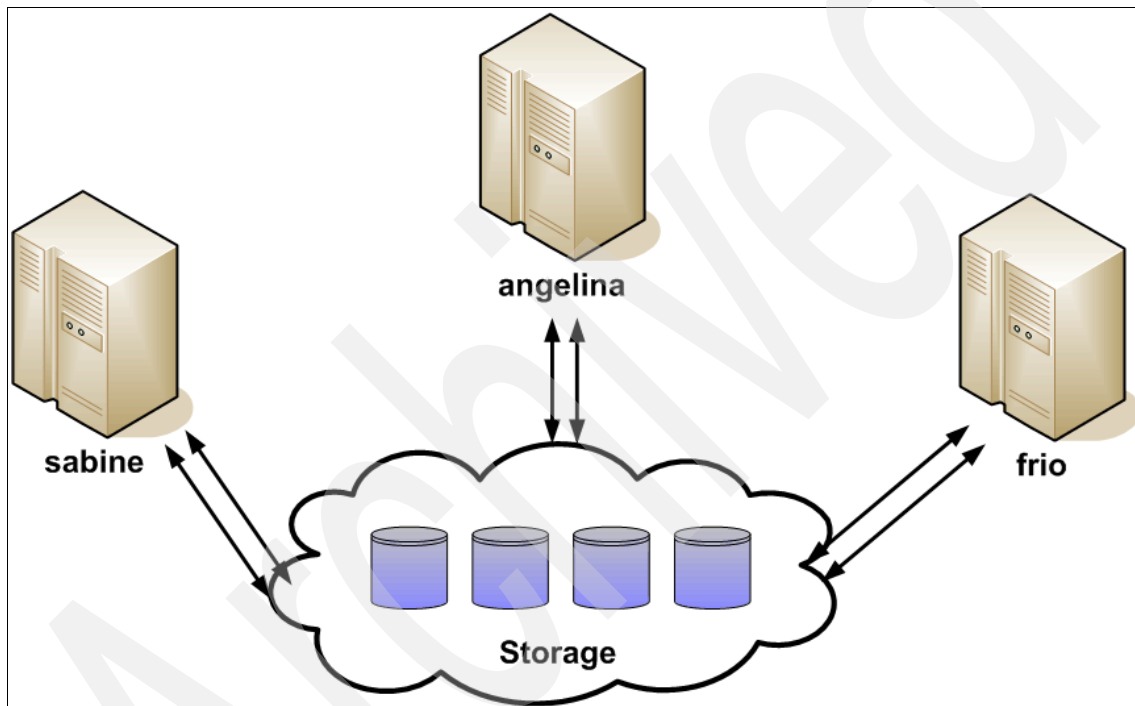


Figure 6-3 GPFS test setup used for this book

Migration considerations

In this chapter, we discuss in general terms the expected planning steps required for migration from an existing production environment based on AFS, DCE/DFS, earlier versions of NFS, or other existing architectures. This material is not meant to act as an exhaustive, all-encompassing guide because every production environment contains customizations and locally developed tools that cannot be addressed within the scope of this book.

The planning phase is the most crucial part of any new software and hardware implementation project. A concise, detailed plan is necessary to ensure that the migration and implementation phases of the project proceed smoothly. This chapter provides detailed information to assist in planning for a reliable infrastructure for NFSv4 deployment. The underlying concepts are complex; however, after they are understood, it will be a simple matter to design and implement the solution.

Take into account the following considerations when beginning the planning phase:

- ▶ Currently available and deployed hardware, software, and applications.
Consider the following questions:
 - Is a current inventory of assets and their usage available?
 - Is a logical overview of the infrastructure available?
- ▶ The organization's planned IT strategy for future expansion:
 - Is centralized user management planned?

- ▶ Business-driven design and implementation issues:
 - Does a need exist to exchange data with other customers or departments?
 - Do currently deployed applications, whether third party or locally developed, need to be taken into account?

We discuss the following topics in this chapter:

- ▶ General migration considerations
- ▶ Types of migrations
- ▶ Hardware planning
- ▶ Individual component considerations
- ▶ NFSv4 user authorization methods
- ▶ Choosing the appropriate file system types

7.1 General migration considerations

A great deal of preparatory work must be accomplished before the migration of data and user accounts from an existing stand-alone, distributed, or enterprise environment can commence. Well-planned migrations are more likely to enjoy success and avoid downtime or a negative effect on user activities. Occasionally, even the most thoroughly planned migration will partially fail due to the unexpected discovery of an existing component that was not accounted for during the planning phase; therefore, it is wise to build a fall-back position into any planning document in order to restore working production systems when necessary.

It is not necessarily the case that old and new architectures are mutually exclusive, or that only one can be active at a given time. You can use numerous *phased migration* options, based on the site's expectations and requirements. We discuss these in greater detail later in this chapter.

It might also be useful to review Appendix B, "Case study: IBM Global Storage Architecture" on page 313.

Project sizing

The most basic requirement for a successful migration is an in-depth understanding of the existing environment. Detailed information is essential to proper project planning and time scheduling activities. The basic planning steps, as well as the time required for the migration of a small, 100-user NFSv3 system with a few hundred gigabytes of locally stored data to NFSv4, are clearly different than an AFS or DCE/DFS cell containing terabytes of distributed data and 30,000 active users; the amount of testing and execution time required obviously are much greater in the latter example.

In addition, it is important to assess the relative importance of the environment to be migrated in the context of the organization's daily operations. The migration will be simplified if the resources in question are not in constant use or can be bypassed for a period of time without incurring a significant business impact. Environments that cannot be made unavailable during the transition represent the greatest challenge, because it might be very difficult to maintain data consistency if users are making changes to files in the old system while the new is being brought online.

7.2 Types of migrations

This section discusses several common strategies that can be used when switching to a new architecture. These basic methodologies can be adapted and mixed, based on the specific organizational requirements involved.

7.2.1 Switch-over migration

A *switch-over* or *all-at-once* movement of user and application services to a new architecture is the most difficult and time-consuming option, but it might be the only viable option in cases where shared data that must be accessible to all users simultaneously is involved.

In general terms, the switch-over migration strategy involves the following steps:

1. Size the migration, based on a detailed analysis of each subsystem (for example, security, data, namespace, and time) that is involved. We present detailed information about each subsystem later.
2. Develop and test migration procedures. Perform this work in an isolated testing environment, using data that closely approximates the data that will be migrated in the live systems. The size of the security registry or files, quantity of data, directory services, and ancillary utilities such as time providers should provide the migration team with a realistic estimate of the time required to move the production environment.
3. Create a fall-back plan. The team must decide what course of action to take if one or more aspects of the actual migration fail. Choices are necessarily limited if the consequences of a partial migration involve significant outages or inconvenience for the user base and customers; options include completing the migration of those components that succeed while rolling back those that fail, or rolling back the entire migration and starting over at a later date. Each situation will be different, based on the level of integration involved and whether the use of a *hybrid* environment is appropriate or even possible.
4. Create a migration time line. This is especially important because the schedule should be optimized to accommodate other projects whose own deliverables might be affected by the migration. Scheduling a major migration, for instance, during a period when manufacturing is expecting the arrival of a large order request might be inadvisable. Difficult scheduling can make a phased migration more appealing, because that strategy might make the overall work simpler to manage.
5. Immediately before commencing the migration, take backups of all running systems that will be affected.

6. Perform the migration. When complete, run tests where needed to determine whether the work was fully successful. Any older systems not transitioned to the new environment should be shut down and left intact if possible for a fixed period after work is complete; this might simplify recovery from latent problems or missed data.

7.2.2 Phased or rolling migration

A *phased* migration is one in which subsets of users or applications are migrated to the new environment. It is often less stressful and simpler to schedule than a single, all at once operation and might also offer the advantage of permitting easier reuse of existing system hardware. It is sometimes referred to as a *rolling* migration, because many scenarios involve transitioning existing services on a rolling schedule.

This strategy requires administrators, with the cooperation of groups or departments, to identify logical sets of users, file systems, or both that can be migrated to the new environment on a rolling schedule. If user account data is involved, the migration requires the simultaneous movement of account management information, any associated namespace entries (such as a file system skeleton), and physical data owned by those accounts in order to be successful. Relationships among groups must be clearly identified so that those who share data are migrated during the same phase.

A rolling migration requires the same basic steps as those described for the switch-over strategy, but the timeline will be different because multiple dates are involved. It may also permit the reuse of additional hardware, because proper planning can result in systems and storage being made available after each migration phase; this hardware can then be upgraded and reconfigured to serve as the basis for the phase.

We provide an example here. The existing environment in this case is a DCE/DFS cell with two fileset servers (A and B), with other functions located on a third server. A new server is added on the network, and appropriate storage, in this case, a SAN, is configured to provide space for the initial group of users to be migrated. Again as an example, these users' files are located on filesets residing on both server A and server B's disks.

First, the phase 1 users are moved using an appropriate method to the NFSv4 namespace. This frees disk space on the existing DFS servers. Next, the administrator uses the DFS command set to relocate those users slated for migration phase 2 to server B's disks, leaving the space on server A unused.

The administrators can then decommission server A from the DCE/DFS cell and, optionally, add it to the growing NFSv4 environment. The process is then

repeated for subsequent groups of users, applications, or both, until all files have been removed from the existing DFS space. Figure 7-1 illustrates this example.

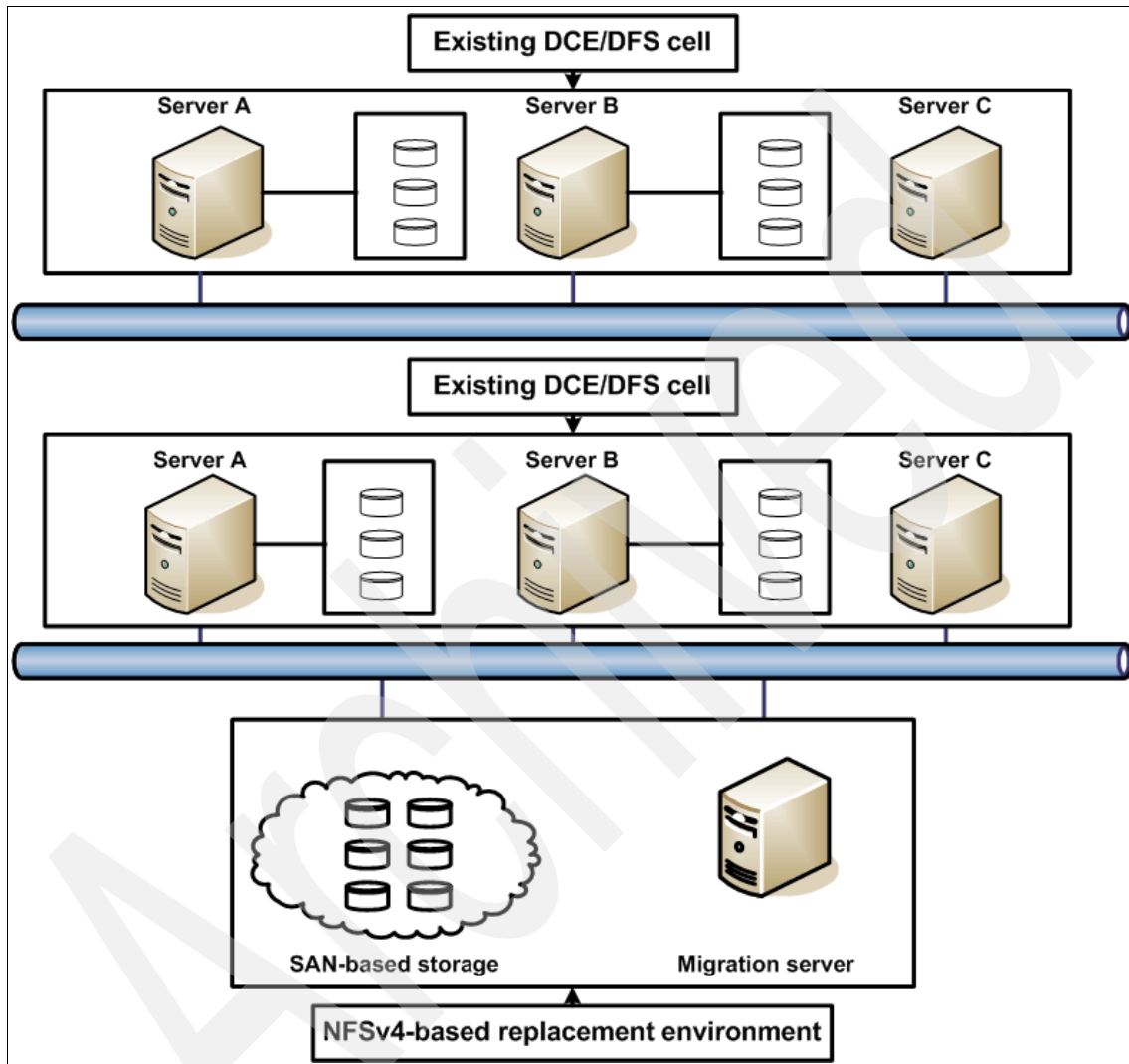


Figure 7-1 Example hardware addition preparation for a phased upgrade

7.2.3 User-by-user or self-managed migration

In this scenario, system administrators prepare a new enterprise environment and provide command line or Web-based migration tools and assistance to the organization's users. Users then migrate their own accounts and data to the new

systems. This strategy has the advantage of giving individual groups and personnel increased flexibility when moving to the new environment and requires only a loose schedule because no fixed transition dates are required.

This scenario involves the following steps:

1. Create the new enterprise file system environment and prepare it for the user community by generating any skeleton directory structures that might be necessary.
2. Develop tools that provide users with the ability to migrate their account and data to the new enterprise environment. Pre-test these tools to ensure their reliability and provide documentation or training as needed.
3. Open the environment to the user community, monitoring their progress on a daily basis while assisting with any difficult situations (for example, application access, and group or user permissions).
4. Establish, again in consultation with the user community, an end date by which all accounts and data must be migrated to the new enterprise environment.
5. Identify and schedule the migration of any data not directly located in users' personal directory structures to the new namespace.

From an overall standpoint, a user-accessible migration script might perform the following tasks:

1. Query the user for their login ID and current password. Note that the latter represents a security vulnerability; therefore, we suggest that encryption techniques be used where available.
2. Using the information captured in step 1, perform an authentication function (for example, `authenticate` or `klog`, `dce_login`) using the existing environment's security subsystem. This ensures that the users have entered their account details correctly and that we have their unencrypted password for later use.
3. Provide the user with a "please wait" message, and then either lock the keyboard or otherwise prevent further data entry until the process has completed.
4. Extract the user's existing account details from the existing environment's security subsystem, and then use these to create an account in the new enterprise environment. Use of the password captured in step 1 eliminates the need to set a random, one-use initial password and simplifies the migration from a user perspective.
5. Recursively copy the user's files and directory structure to the appropriate home directory in the new enterprise architecture's file system, ensuring that they are properly owned by the newly created account.

6. Emit a success or failure message based on the results.
7. Optionally, lock the user's account in the existing environment's security subsystem to prevent the user from modifying data in both the existing and new environments.

Manual intervention by system administrators or other personnel might be necessary when failures occur or special cases (for example, existing accounts with large quotas, symbolic links, or other unusual circumstances) are encountered. Ideally, the migration application should log its activities to a known location in order to simplify troubleshooting activities.

7.3 Hardware planning

A fundamental decision must be made whether to reuse existing hardware or purchase one or more new systems for use as a test bed, and later as servers for the new environment. Consider the following basic strategies:

- ▶ An in-place upgrade using only existing hardware, during which the old environment is completely decommissioned and replaced. Unless additional disk hardware is acquired, this option might require the reformatting of volumes currently used for certain file systems (for example, DFS or AFS) in order to establish the basic NFSv4 namespace. This option is only useful in the context of the switch-over or “all at once” migration strategy, because the existing architecture is made completely unavailable during the upgrade process.
Recovery time in the case of a failed migration attempt frequently will involve restoration of the most recent backups; this should be factored into both the hardware decision making and project scheduling processes.
- ▶ Similarly, it might be possible to run both existing and new services concurrently on existing hardware, producing a hybrid environment suitable for a rolling or user-by-user migration strategy. This option might require additional disk space if, for example, concurrent access to both NFSv4 and DFS or AFS data is required.
- ▶ The purchase of one or more systems for use in creation of the new NFSv4 namespace, with subsequent reuse of existing hardware after the basic services have been established. This option might work well with the rolling upgrade, but is also viable for other situations.

- ▶ The creation of a completely new server environment. This option supports multiple migration strategies and provides increased flexibility, because it eliminates the need to reallocate hardware supporting existing file systems. Additionally, existing systems will still be available if a migration attempt fails, resulting in a shorter recovery time.

Address the following general concepts during the hardware decision process:

- ▶ Is the existing hardware capable of supporting sufficient disk space and other resources to support the new environment?
- ▶ Is it possible to maintain the production environment on available hardware while services are added to accommodate NFSv4 or another enterprise file system? Will such activities cause a performance degradation or other problems for the user base, and can the old and new services coexist successfully?
- ▶ Is it more or less cost effective to perform a migration in place using existing hardware, and is it more efficient to perform both hardware and software upgrades simultaneously?

Tip: In the test environment used to illustrate examples in this book, we found it beneficial to exploit the abilities of the IBM POWER5™ architecture with AIX 5L. By using logical partitions on the POWER5 platform, we created NFS servers and clients to test various migration strategies. Although the test environment was relatively simple compared to a production migration, the concept of using LPAR and Micro-Partitioning™ server technology might significantly reduce migration costs from both a test environment and production environment standpoint. This technology allowed for a much larger test environment than the physical number of machines that were used.

7.4 Individual component considerations

This section provides additional details regarding the steps necessary to migrate each major component from an existing environment to a newly installed enterprise architecture.

7.4.1 Security

NFSv4 introduces many substantial changes to the protocol. For example, the design of NFS predates the widespread adoption of the World Wide Web; the NFS protocol was originally designed for use in local area networks. With the advent of internetworking, a greater need has arisen for the use of distributed file systems such as NFS in wide area networks. When NFS is used for distributed file systems across wide area networks, the security limitations in pre-Version 4

implementations become apparent. Addressing existing security limitations in NFS is the most significant area of change introduced by Version 4.

The topic of security can be very broad and far-reaching. A discussion about security measures includes items in the following categories:

Physical security	Measures taken to control physical access to a facility or resource. Padlocks, fences, guards, and dogs are examples of physical security measures.
Personnel security	Measures taken to help ensure that the people who are granted access to secured resources are reliable and are not likely to compromise the security of those resources. Security clearances and photo ID badges are examples of personnel security measures.
Information security	Measures taken to protect important information from unauthorized disclosure, tampering, or destruction. Passwords, encryption, and file access permissions are examples of information security measures.

This material concentrates on the area of information security, because NFSv4 is designed to enhance the process of data sharing. Although it is impossible to properly protect information resources without also implementing physical and personnel security measures, a discussion of these topics is beyond the scope of this book.

7.4.2 RPCSEC_GSS security flavors

NFSv4 uses the Sun remote procedure call (RPC) protocol to communicate over the network between the client and the server. The IBM implementation enables you to use three different RPC security flavors:

- ▶ Basic UNIX security (AUTH_SYS, also known as AUTH_UNIX)
- ▶ Diffie-Hellman security (AUTH_DH, also known as AUTH_DES)
- ▶ RPCSEC_GSS security as defined in RFC 2203

7.4.3 RPCSEC_GSS protection levels

When using RPCSEC_GSS security, three levels of protection are available that can be applied to RPCs as they are transmitted over the network between server and client:

Authentication	Validates the identity of RPC sender.
Integrity	Validates that the contents of the RPC were not changed during transmission (also includes authentication).

Privacy Prevents unauthorized viewing of data while it is in transit between client and server (also includes authentication and integrity).

Keep in mind that each increasing level of protection entails a performance penalty. We recommend that organizations start with the minimum level that meets data protection requirements, increasing levels as needed while assessing the impact of each change on performance and usability.

The NFSv4 standard (RFC 3530) requires that NFS implementations support three different RPCSEC_GSS mechanisms:

- ▶ Kerberos V5 (RFC 1964)
- ▶ SPKM-3/LIPKEY (RFC 2847)
- ▶ SPKM-3 on its own (RFC 2847/RFC 2025), for situations where the initiator (the client) is anonymous or has its own certificate

The NFSv4 implementation on AIX 5L V5.3 uses Kerberos V5 security. There is no support for SPKM/LIPKEY at this point in time.

The IBM implementation of Kerberos V5 offers several forms of encryption, among which are single Data Encryption Standard (DES) and triple DES. Triple DES encryption provides the best protection, but single DES results in improved performance and interoperability.

7.4.4 User identity management options

UNIX implementations typically use a 32-bit integer to identify users and groups. These integers are referred to as UIDs for users and GIDs for groups. User and group ownership for system processes and file system objects are maintained in UID/GID form. NFSv2 and NFSv3 also use these UIDs and GIDs to identify users and groups. Users typically do not work directly with numeric IDs; instead, they work with text user and group names that are easier to associate with an actual individual or group. When presenting information about process and file ownership, the system translates a numeric ID into its associated name. The relationship between the names and the IDs is maintained in a user registry, which can be standard UNIX or LDAP.

Standard UNIX user registry

In standard UNIX, user name-to-ID mappings are contained in the `/etc/passwd` file, and group name-to-ID mappings are stored in the `/etc/group` file. All but the smallest organizations use a shared user registry rather than maintain separate `/etc/passwd` and `/etc/group` files on all hosts; otherwise, it is impossible to ensure that ID-to-name mappings are consistent across all machines. Clients using data

stored on an NFS server must use the same identifier to represent the same user or group in order to maintain consistent file ownership.

For example, if UID 100 is Joe on one NFS client and Mary on another client, NFS files created by Joe from the first client will show as being owned by Mary on the second client, and vice versa. To avoid this, individual `/etc/passwd` and `/etc/group` files must be kept in sync across all clients that access data on a common NFS server. This can be a very expensive and error-prone task.

LDAP user registry

Directory services based on the Lightweight Directory Access Protocol (LDAP) can also be used to maintain user and group identities. LDAP-based directories are typically more scalable and secure than `/etc/passwd` or Network Information Service (NIS). RFC 2307 describes an LDAP schema that provides NIS-like functionality. IBM Tivoli Directory Server is the IBM implementation of LDAP-based directory services. As of Version 5.1, Tivoli Directory Server supports RFC 2307 user and group identification. More information is available about the Tivoli Directory Server on the IBM Web site at:

<http://www.ibm.com/software/tivoli/products/directory-server>

7.4.5 User and group identities and NFSv4

NFSv4 handles user and group identities in a different manner than previous versions. NFSv2 and NFSv3 pass UIDs and GIDs between the client and server, while NFSv4 passes string names in the form `user@nfs_domain` or `group@nfs_domain`. For purposes of this book, entity mapping is described under three conditions:

- ▶ Single NFS domain using AUTH_SYS security
- ▶ Single NFS domain using Kerberos security
- ▶ Multiple NFS domains

Two sample operations represent each case:

- ▶ Creating a file
- ▶ Requesting file ownership

7.4.6 RPCSEC_GSS user authentication using Kerberos

How does NFS use Kerberos? When an NFS client and server use Kerberos V5 authentication, the client and server establish a security context for NFS requests. The security context is a data structure that indicates that the client and server have completed a mutual authentication procedure. The context also contains encryption keys used for protecting exchanged data, if such protection

has been requested. The security context has a lifetime and might need to be refreshed by the client from time to time. The client and server each maintain a cache of security contexts, one context per user/host combination.

7.4.7 User accounts and authentication resources

User account data often represents a difficult decision when migrating an active environment to a new architecture, because tools for automating this step might or might not be available, or might require extensive customization for successful use in a particular environment.

Managing authentication data

Basic information such as a login ID, password, UID, and group data are the foundation of an authentication/authorization subsystem. As noted earlier, these can be found in a flat ASCII file such as a UNIX `/etc/passwd` file or in a structured database such as LDAP. Along with the basic authentication data, existing accounts might or might not include components such as:

- ▶ Access control lists (ACLs) that provide per-user or per-group authorization for resources such as files, directories, or devices
- ▶ Group membership lists
- ▶ Account expiration and creation dates, or last-access records, or both
- ▶ A type of “validity” bit, such as the `acctvalid` entry in a DCE registry or the shell value in an `/etc/passwd` file, which is often set to `/bin/false` to disable a given user’s ability to log in
- ▶ Time-based login or connection limitations (for example, users who are permitted to log in only between the hours of 8 a.m. and 6 p.m.)
- ▶ Personally identifying user information, such as office location, given name and surname, telephone, or pager numbers

For example, an entry in a UNIX `/etc/passwd` file might include the data shown in Example 7-1.

Example 7-1 Typical UNIX user account information

```
> grep gelaotis /etc/passwd
gelaotis:!:8512:11:Vasilios Gelaotis:/users/g/gelaotis:/bin/csh
```

However, an AFS or DCE/DFS user account includes a great deal more data, reflecting the increasing flexibility and complexity of the environment. Typical user accounts might look similar to those shown in Example 7-2 on page 174 and Example 7-3 on page 174.

Example 7-2 Typical AFS user account and group information

```
# /usr/afs/bin/kas examine user2
Password for user2:
User data for user2
  key (0) cksum is 4088618827, last cpw: Tue Aug 16 11:05:15 2005
  password will expire: Sat Feb 18 10:05:15 2006
  10 consecutive unsuccessful authentications are permitted.
  The lock time for this user is not limited.
  User is not locked.
  entry expires on Sun Jan  1 06:50:02 2006.  Max ticket lifetime 25.00
  last mod on Thu Aug 18 12:53:14 2005 by admin
  don't permit password reuse
#
# /usr/afs/bin/pts examine user2
Name: user2, id: 215, owner: system:administrators, creator: admin, membership:
2, flags: S----, group quota:
20.
```

Example 7-3 Typical DCE user account data

```
%unix1> dcecp -c user show gelaotis
{fullname {Gelaotis, Vasilios E.}}
{uid 95556}
{uuid 00017544-48f5-21d3-8500-02608c2f5013}
{alias no}
{quota unlimited}
{groups subsys/dce/foo-admin foo-ibm foo-all build change-team}
{acctvalid yes}
{client yes}
{created ../../test.austin.ibm.com/cell_admin
1999-08-02-12:12:43.000-04:00I-----}
}
{description {Gelaotis, Vasilios E.}}
{dupkey no}
{expdate 2006-04-29-20:00:00.000-04:00I-----}
{forwardabletkt yes}
{goodsince 1999-08-02-12:12:43.000-04:00I-----}
{group dce-all}
{home ../../test.austin.ibm.com/fs/u/gelaotis}
{lastchange ../../test.austin.ibm.com/gelaotis
2005-06-27-15:49:01.000-04:00I----}
-}
{organization none}
{postdatedtkt no}
{proxiabletkt no}
{pwdvalid yes}
{renewabletkt yes}
{server yes}
```

```
{shell /:/u/gelaotis/bin/tcsh}  
{stdtgaauth yes}  
{usertouser no}  
nopolicy  
>
```

As these examples show, the quantity and nature of user account data stored in the original environment varies greatly across the various authentication schemes. This example also shows that information may have a single source, such as the AFS databases and DCE registry, or multiple providers, as is the case when separate UNIX password and group files are involved.

A general, though not exhaustive, list of possible account locations include:

- ▶ /etc/passwd and /etc/group.
- ▶ The /etc/shadow file (if shadow passwords are in use).
- ▶ One or more Network Information Service (NIS) databases.
- ▶ An AFS kaserver database.
- ▶ A DCE/DFS security registry.
- ▶ An LDAP database.
- ▶ An IBM Network Authentication Services, Sun Microsystems Sun Enterprise™ Authentication Mechanism™ (SEAM), or MIT Kerberos V5 key distribution center (KDC) database.
- ▶ A Microsoft Windows Active Directory database (note that this example involves a mixture of LDAP and Kerberos data).

The situation can be further complicated if the existing infrastructure makes use of multiple authentication methods. For example, some accounts might use the standard /etc/passwd access method while others reside in an AFS authentication database or DCE registry.

For example, the only commercially available tool known to exist that automates the operation of importing general /etc/passwd user information into an IBM Network Authentication Service or Kerberos V5 database is the AIX **mkseckrb5** utility. A similar situation exists in respect to migrating AFS users into a DCE/DFS or generic MIT Kerberos database. DCE, however, includes the **passwd_import** tool. This tool enables a DCE registry to import accounts from standard UNIX /etc/passwd and /etc/group files. Other tools might be available, but often require extensive customization on a site-by-site basis.

Any tools written to manage these operations must be designed to accommodate the often ambiguous or incomplete nature of existing account structures. For example, depending on the site's existing policy, an AFS/DCE user's account or

/etc/passwd entry might or might not contain personally identifying data. In other cases, sites have customized the appearance of passwd file to include departmental or other data through the General Electric Common Operating System (GECOS) field. The account data in Example 7-4, for example, shows a user with a GECOS field containing the user's name, group, and telephone number.

Example 7-4 Use of expanded data in the GECOS field

```
gelaotis!:8512:11:Vasilios Gelaotis, ABC Group, 555-1212:/users/g/gelaotis:/bi  
/csh
```

The same caveat applies to other authentication methods. Additional care is required, for instance, if a DCE/DFS site has used the security registry's fullname field to store the user's name, telephone number, or other data in a manner similar to that shown in the previous passwd file entry.

In this case, extra code should be included in the site's migration scripts to parse the passwd file's "User, Group, Phone" format, which is hopefully consistent across all accounts, into discrete entities in order to insert it into the proper fields in an LDAP or other database schema.

Migrating existing user passwords is often impossible due to the encrypted nature of password storage. Therefore, migrated users might be faced with new or one-time-only passwords in the new authentication environment, necessitating further development of an initial login utility that does not require the user to know their current password, or a secure method of disseminating new passwords to the user base.

Note that multiple export/import tools might be required, depending on the heterogeneity of the site's existing environment. One tool will be required for Microsoft Windows-based accounts, another for passwd-file based UNIX users, another for AFS, and so on.

Access control lists

Some sites make extensive use of access control lists (ACLs) to manage access to files, directories, and other resources. Others rely simply on group membership and the standard UNIX **chmod** command, or on basic Windows file permissions. ACLs are not part of the standard UNIX passwd/group environment, which uses a basic user, group, others file permissions model to control access. Sites considering a migration from a traditional files-based UNIX environment will have less work to do to migrate their environment, but will still need to prepare the NFSv4 environment appropriately.

The problem of ACL migration is complicated by differences in implementation across platforms and file systems. It is not always a simple task to create a one-to-one mapping between ACLs on the existing environment and those found in, for example, NFSv4.

These differences among ACL implementations across various file systems can make it simpler to abandon existing schemes and start fresh within the new enterprise architecture. If this is not desirable, administrators will have to acquire or develop procedures to extract existing ACL information from their current environment for use in establishing an NFSv4-compliant ACL environment on the new environment. We demonstrate simple examples of how this can be accomplished in our example migration chapters.

Sizing the security migration

Next, we describe the fundamental steps in sizing the migration of security services to a new architecture. The basic idea is to determine the amount of time required to migrate the entire security architecture to its new home in a worst-case scenario. Perform the following steps:

1. Determine the number of accounts to be migrated.
2. Establish the account sources (for example, passwd file, DCE, NIS, AFS, or Windows).
3. Configure a test environment for the new architecture, such as NFSv4. Ideally, this should be accomplished using an isolated network to avoid the possibility of disrupting running production systems.
4. Perform initial migration testing using, if possible, a copy of the live data because this can produce a more accurate representation of the time involved.

During this phase, establish the expected time necessary to develop scripts and other tools to be used for automated account migration.

5. Test the scripts, along with any tools or procedures developed in the other steps outlined earlier, in the isolated test environment.

In the case of a switch-over migration, the time required to migrate the live security data from the existing environment to the new systems is a subset of the total time necessary to accomplish the full migration. The migration team should estimate the time requirement based on earlier testing, because this number must be added to the time required to transfer all user and application data, along with directory, time, backup, or other services to the new infrastructure.

7.5 NFSv4 user authorization methods

Authorization is used to control access to either client hosts or client users.

Three choices are available for controlling client user access to files and directories:

- ▶ Standard UNIX permissions
- ▶ AIXC ACLs
- ▶ NFSv4 ACLs

7.5.1 Choosing a user authorization method

An accurate assessment of a site's requirements for controlling access to data prior to deciding which user authorization method to implement is needed. Are the data access requirements simple or complex?

Standard UNIX permissions allow access control for only three identities:

- ▶ The owning user
- ▶ The owning group
- ▶ Everyone else

If that level of granularity is not sufficient to meet the organizations's access control requirements, select an enhanced access control list (ACL) option. For example, the use of standard UNIX permissions is insufficient if one group requires write access to a portion of data, while one or more other groups require read-only access, and remaining users have no access at all.

AIXC ACLs or NFSv4 ACLs can be used if standard UNIX permissions do not meet current requirements. An appropriate file system structure (for example, JFS2 or GPFS) must be selected if NFSv4 ACLs are selected.

Do not use AIXC ACLs if the requirements include one of the following items:

- ▶ Non-AIX NFS clients that need the ability to manipulate ACLs for data on an NFS server. AIXC ACLs are only supported on AIX 5L.
- ▶ A finer granularity level of access control than AIXC ACLs support is required. For example, AIXC ACLs do not provide a way to create a directory where users can create files but not delete them after they are created.

Note: The choice of AIXC ACLs over NFSv4 ACLs should make sense in environments that already have experience with the AIX ACL model and are running predominantly AIX 5L platforms. Otherwise, we recommend the use of NFSv4 ACLs as the default choice due to its compliance with open standards and to minimize potential interoperability issues with non-AIX platforms.

7.5.2 Other user authorization considerations

In this section, we discuss other user authorization considerations.

Maintaining access control lists

Maintaining access control lists is more complicated than maintaining standard UNIX permissions. A clear understanding of ACL evaluation is required before an ACL can be correctly constructed. (See 5.4.4, “NFSv4 ACLs: Permissions scenarios” on page 110 and Figure 5-10 on page 113 for more information about how NFSv4 ACLs are evaluated.)

Depending on the level of sophistication of the user community, it might be necessary either to educate them on the proper maintenance of ACLs or set up ACL inheritance to eliminate any need for user-managed ACLs. (See 5.4.3, “NFSv4 ACLs: ACL inheritance and umask” on page 103 for more details about how to set up ACL inheritance.)

What if a security policy is implemented to prohibit end users from changing permissions, thus keeping that task under the control of designated system or data administrators? The system does not accommodate this by default. Files created by a user are owned by that user, and a file’s owner can always change its permissions. In this case, it is necessary to devise some way to make sure all files are owned by an administrator account. One way to do this is to run a periodic cron job that changes ownership of all files to that account. This leaves a window of time when a user can change permissions on a newly created file, but this can be remedied by the job that changes ownership. It can also ensure that file permissions are set appropriately after the ownership has been changed.

From this discussion, you can see that it might be necessary to implement additional administrative controls in addition to those that are provided by default directory services.

Existing directory services

The term *directory service* is often used in an ambiguous or confusing manner, reflecting its varying usage in certain contexts. In DCE terms, the Cell Directory Service (CDS) provides both the structure of the DFS namespace and a locator

service for applications that are seeking the current IP address and endpoint for a given server process. CDS also provides ACL storage for DCE/DFS objects, and therefore might be relevant if a site wants to migrate an existing ACL structure to an NFSv4 or other enterprise storage architecture. The Domain Name System (DNS) is a directory service that maps host names, such as www.ibm.com, to one or more Internet Protocol (IP) addresses.

In other contexts, a directory service might refer to a name-to-electronic mail address mapping system or one that provides end-user access to an employee database. The X.500 and Lightweight Directory Access Protocol (LDAP) architectures are often used to provide these and other services. IBM Network Authentication Service, for example, can be used in conjunction with the traditional Kerberos V5 use of flat file databases to provide extended storage of user account data similar to that found in the DCE CDS directory service.

When planning a migration from a DCE/DFS environment, it must be determined whether CDS is being used for purposes other than those required by DFS. These can include locally written client/server applications, products such as the Encina or Distributed CICS® transaction processing systems, certain Object Request Broker (ORB) systems, IBM InfoPrint system, and others. If other applications make use of CDS, it will be necessary to maintain basic DCE services even after the DFS components have been decommissioned, unless the applications in question can be redesigned to use another directory service. Further discussion of the topic of application migration is outside the scope of this document, but for additional details, see *DCE Replacement Strategies*, SG24-6935.

Note that CDS is dependent on the DCE remote procedure call (RPC) mechanism, and a minimal DCE cell also requires an active security daemon in order to function. This situation might require certain users to maintain accounts, for example, in both the DCE registry and Kerberos V5 system used for access to NFSv4.

AFS does not offer directory services outside its role as a provider of file system access and authentication, so the previous paragraphs are inapplicable with respect to a migration from AFS to NFSv4 or another architecture.

Other directory providers

Two options are available for those sites currently running X.500 directory services:

- ▶ Maintain both the existing service and LDAP. This might be applicable in cases where it would be too labor intensive to move the X.500-resident data into an LDAP environment, or there is no conflict between data stored in the two locations.

- Make use of an automated tool to migrate the existing X.500 data into the LDAP space. In this case, great care should be taken to ensure data consistency; if both directory services contain, for example, personnel data, it will be necessary to ensure that no records are accidentally overwritten.

7.5.3 NFSv4 host identification

This section discusses two forms of host identification: basic identification through IP addresses and host names, and Kerberos identification through a machine principal.

Basic host identification

An NFSv4 server identifies client hosts by the IP address given in the RPC packets. The NFS server turns this IP address into a host name through the host resolver, which can either get its information from the Domain Name System (DNS), Network Information Service (NIS), or the local `/etc/hosts` file.

Kerberos host identification

Kerberos authentication uses a unique identifier called a *machine principal* to identify hosts. The machine principal is established when configuring a host into a Kerberos realm. The machine principal name is the fully qualified host name prefixed with `host/` (for example, `host/nfs402.itsc.austin.ibm.com`).

Another way that Kerberos indirectly identifies a host is through the NFS *service principal*. (It is the identification of the NFS service running on the host.) The service principal name is the fully qualified host name prefixed with `nfs/` (for example, `nfs/nfs402.itsc.austin.ibm.com`). NFS clients using Kerberos authentication identify NFS servers through this service principal.

7.5.4 NFSv4 host authentication

NFS servers always identify client hosts through IP addresses and host names, regardless of the authentication method used. The value added with NFSv4 is that when Kerberos authentication is the only allowed security method for an exported directory (see the following host authorization section), the NFS client session must be properly authenticated before gaining access to any of the data in that directory.

Think of NFSv4 authentication of clients being mostly at the user level rather than at the host level.

Kerberos does authenticate NFS server identities to the clients via the NFS service principal. (See the Kerberos host identification discussion in the previous section.)

7.5.5 NFSv4 host authorization

Host authorization in an NFS context means controlling which NFS client hosts can mount exported directories from the NFS server. This is accomplished in AIX 5L through a combination of the `/etc/exports` file and the **exportfs** command.

Exporting directories from an NFS server is still fundamentally the same in NFSv4 as it was in NFSv3. The main difference is that NFSv4 has added the new security-related options shown in Table 7-1.

Table 7-1 New `/etc/exports` security options for NFSv4

Option	Description
vers	Controls which version of NFS mounts are allowed. Possible values are 2, 3, and 4. Versions 2 and 3 cannot be enforced separately. Specifying version 2 or 3 allows access by clients using either NFS protocol versions 2 or 3. Version 4 can be specified independently and must be specified to allow access by clients using version 4 protocol. The default is 2 and 3.
sec	Controls which security methods are allowed. Possible values are: <ul style="list-style-type: none">▶ sys: UNIX authentication.▶ dh: DES authentication.▶ krb5: Kerberos, authentication only.▶ krb5i: Kerberos, authentication and integrity.▶ krb5p: Kerberos, authentication, integrity, and privacy.▶ none: Allow mount requests to proceed with anonymous credentials if the mount request uses an authentication flavor not specified in the export. Otherwise, a weak auth error is returned. By default, all flavors are allowed. In the absence of any sec option, sys (UNIX authentication) is assumed.

The `sec` option is unique in that it can appear more than once in the exports definition for a directory. This allows different `ro`, `rw`, `root`, and access options to be specified for the different security options. For example, hosts using the `sys` security method might only be allowed read access, while hosts using the `krb5` security method might be allowed read and write access.

Note: The same security option cannot be specified in more than one `sec=` stanza in a single exports definition.

A sample `/etc/exports` line with the new NFSv4 security options is:

```
/exports -vers=3:4,sec=krb5:krb5i:krb5p,rw,sec=sys:none,ro
```

For more details about exporting directories, see *AIX 5L Version 5.3 System Management Guide: Communications and Networks*, SC23-4909, the **exportfs**

command in *AIX 5L Version 5.3 Commands Reference* (<http://publib.boulder.ibm.com/infocenter/pseries/index.jsp>), and the `/etc/exports` file in *AIX 5L Version 5.3 Files Reference*, SC23-4895.

7.6 Choosing the appropriate file system types

AIX 5L supports several different file system types, including:

- ▶ Journaled file system (JFS)
- ▶ Enhanced journaled file system (JFS2)
- ▶ JFS2 with extended attribute format version 2 (EAv2)
- ▶ General Parallel File System (GPFS)

The file system chosen will be dictated by the authorization method selected earlier. If NFSv4 ACLs are required, the choice is limited to JFS2 EAv2 or GPFS. This does not mean systems cannot use the other file system types; the restriction only applies to the file systems where NFSv4 ACLs are in use.

7.6.1 Backup systems

Many existing file system products use proprietary backup systems. These services might produce backup media that is unreadable by other means. Any migration to a new enterprise architecture must take the subject of archival data management into consideration.

AFS includes backup and restore commands that dump data into *volume sets*. Additionally, it makes use of one or more Backup Tape Coordinator (butc) processes and an internal database to maintain a list of backups, volume sets, and other data used for restoration purposes. DCE/DFS provides a similar backup system based on the AFS model. If these utilities have been used for the creation and management of regular backup media, it might be necessary to maintain some level of support for existing file system, because it is not possible to restore AFS or DFS dump sets using other software. The organization's backup retention policy can be used to determine the amount of time that the existing environment must be maintained.

This does not apply if a file system-independent product, such as IBM Tivoli Storage Manager, is in use, because it will be able to restore data archived on the existing file systems architecture. The same applies in cases where sites use common tools such as `cpio`, `dump`, or `tar` for daily backup activities.

7.6.2 Time services

Time coordination is critical to the success of distributed file access, because access and revision time stamps must be accurately recorded in order to maintain data consistency. Incorrect file time stamps can mislead users or application programs that rely on accurate recording to maintain checkpoint files, backups, or other time-sensitive data. Some existing file system products include a time coordination subsystem or utility in order to address this problem. Alternately, most, if not all, of these products support the use of the Network Time Protocol (NTP).

Additionally, both IBM Network Authentication Service and MIT Kerberos V5 require time synchronization between server and client in order to maintain user credentials and prevent “replay attacks” mounted by malicious users. By default, times must be accurate to within 5 minutes or client requests will be rejected by the server. DFS requires even tighter control, with all clocks synchronized to within 30 seconds of each other.

Example time services

AFS makes use of the Cache Manager to coordinate clocks with server systems within the cell. Sites that make use of NTP disable this feature by running the **afsd** daemon with the **-nosettime** switch, which disables the built-in synchronization.

DCE provides the Distributed Time Service, or DTS, for the same purpose. Many sites use DTS as a global time provider, while others standardized on NTP as a freely available alternative.

Current releases of Microsoft Windows support the use of the Windows Time Service, which is based on an implementation of the Simple Network Time Protocol (SNTP) as detailed in RFC 1769.

Implementation considerations

Before establishing a new enterprise file systems architecture, an appropriate time coordination service should be identified and implemented across all installed systems, both client and server. Only one time service should be in use in a given environment in order to avoid conflicts between providers.

System times should never be altered manually if either AFS or DCE/DFS are in use because both rely on the Kerberos protocol, which is very sensitive to such alterations. Time provider services are designed to make slow, gradual changes to system times when drift occurs to avoid disrupting file time stamps and other activities.

7.6.3 User data

Migration of an existing directory structure to a new namespace can represent an opportunity to replace an outdated hierarchy, or to establish rules for the overall appearance of the file system. Should the existing structure be maintained, or does the migration represent an opportunity to restructure it to improve manageability, performance, or other characteristics? Many organizations use a relatively flat directory structure. However, in large environments, this can degrade performance due to large directory sizes. In order to eliminate this problem, a *hashed directory structure* can be used. This involves creation of a deeper hierarchy, minimizing the size of each directory and thereby decreasing traversal time. Hashes of various depths can be used, but most sites use no more than two levels.

Example 7-5 shows an example of a two-level hashed directory structure.

Example 7-5 Example of a hashed directory structure

```
unix%> ls -RAC /home | more
/home:
a
/home/a:
a b c d e f g h i j k l m n o p q r s t u w x y z

/home/a/a:
aardvark aaron

/home/a/a/aardvark:

/home/a/a/aaron:

/home/a/b:
abner

/home/a/b/abner:

/home/a/c:
```

The hashed tree can also be organized by department or group, perhaps using a hierarchy such as that shown in Example 7-6 on page 186. Note that the use of long directory names such as “administration” is discouraged for performance reasons; shorter names improve directory lookup times.

Example 7-6 Another hashed directory structure

```
unix%> ls -RAC /home | more
/home:
a

/home/a:
accounting  administration

/home/a/accounting/:
aardvark  aaron

/home/a/accounting/aardvark:

/home/a/accounting/aaron:

/home/a/administration:
abner

/home/a/administration/abner:
```

The same question should be asked regarding file permissions and ACLs. Is the existing scheme adequate, or does it require review? This might also be an area of concern because NFSv4 ACLs, as noted earlier, do not map directly to those used in NFSv3 or other file systems. It might be difficult to write scripts that will correctly copy an existing ACL structure from the existing environment, especially if complex ACLs were in common use.

Symbolic links

The presence and extent of use of symbolic links in an existing directory structure will significantly affect data migration sizing and complexity. If AFS is in use, the existing namespace begins at the /afs/<cell_name> root. DFS directories will appear using either /:/ or /.../<cell_name>, depending on which convention is in use at the site. Therefore, a symbolic link based on a fully qualified directory name might appear as shown in Example 7-7; both links point to the same file.

Example 7-7 Symbolic link based on a full qualified directory name

```
# pwd
/:/home/joe
#
# ls -l
total 0
-rw-r--r--  1 mary    sales          0 Aug 22 10:01 file
lrwxrwxrwx  1 joe     sales      25 Aug 25 14:15 file_symlink.txt ->
/:/home/anthony/file1.txt
```

```
lrwxrwxrwx 1 joe sales 50 Aug 25 14:37 file_symlink2.txt ->
/.../itsc.austin.ibm.com/fs/home/anthony/file1.txt
#
```

Migration of this directory tree to a new namespace beginning with /nfs obviously causes breakage, because neither /:/ nor /.../itsc.austin.ibm.com/fs map to a legitimate location in the new file system. You can achieve a resolution of this problem either by changing existing symbolic links as they are encountered, or by creating a link in the NFS space mapping to the old root, for example:

```
ln -s /nfs /:/ or ln -s /nfs /afs/<cell_name>
```

The latter solution can be left in place until users and administrators convert existing links to use the /nfs root, or it can remain active indefinitely. Any symbolic links created after migration is complete should refer to the /nfs root in order to avoid exacerbating the problem.

Data migration sizing

Time and planning requirements for movement of physical data to the new environment is necessarily based on the amount of data, the migration method (for example, switch-over, rolling, or user-by-user), and the method by which it is to be moved. Consider the following information:

- ▶ If the source file systems will be NFS-mounted on the destination server, the copy speed depends on network bandwidth and load. Prior to migrating data, test copies from source to destination might help to establish average rates that can then be used to predict the total amount of time required.
- ▶ If tapes are to be used, the total data copying time will be the amount of time required to create the media on the source servers, transport it to the destination, and restore it on the destination servers. Again, advance testing will be useful in generating predictive data regarding total time requirements for this step.
- ▶ If replication is required, additional time must be scheduled in order to create and synchronize the replicas.

It is important to generate time estimates well before the final migration date is decided on (except in the case of a user-by-user scenario, which generally should require no system outage). The decision whether to migrate on a given date can be predicated on the total amount of time estimated for the overall migration to occur. If estimates indicate that only a few hours are necessary, scheduling a single planned outage outside peak business hours might be an option. Longer time estimates need to be evaluated against business commitments and the overall impact on the user base. After the total time has been estimated, all options should be re-examined to determine if the correct migration strategy has been selected.

Switch-over migration

If the intention is to migrate all data and user accounts simultaneously, it will be necessary to create a full backup of all data immediately prior to commencement of migration activities. All users must be barred from logging in during this process, and all files slated for movement must be closed in order to eliminate the possibility of inconsistency. It is then a matter of moving the data to newly created NFSv4 file systems.

Phased

This option can be ideal in situations where immense amounts of data are involved because it offers the option of planning multiple, short, off-peak outages for specific groups. Here again, a full data backup of all affected user accounts is mandatory prior to moving any user or other files, and the designated user groups must be prohibited from accessing the systems throughout the migration period.

The phased migration can introduce additional complexity if users have data or other files that are shared among groups or departments. If group A shares group B's files, both groups must be moved during the same phase in order to maintain consistency regarding the shared data. Likewise, movement of shared databases or other resources must be carefully coordinated to ensure such shared information remains accessible throughout the migration period.

User-by-user

This is by far the most flexible option available and additionally has the advantage of producing the smallest impact on running production systems because no downtime is required. Because there might be no way to predict the exact time a given user performs the migration (and when their last backup occurred), the method used for migrating data from the existing namespace to its new location must ensure that the process completes successfully and should include measures to either lock copies of files located in the old namespace or move them to an archive location to prevent modification.

Again, data not located in user directories must be scheduled for migration in cooperation with those users requiring access to it in order to ensure the appropriate levels of availability. Data consistency issues might arise if a group of 10 accounts require access to a shared file, but 5 are moved while the others continue to use the existing copy. Scheduling, user education, and a detailed understanding of shared file use and access requirements will assist the migration team in planning for situations such as those described here.

Migration scenarios

This chapter introduces the scenarios that we demonstrate in this book. Numerous implementation and migration permutations exist, and it is impossible to discuss every one of them. Therefore, for demonstration purposes, we selected the scenarios that seem most likely to be encountered by migrating customers.

We demonstrate the following migrations:

- ▶ Migration from AFS to an NFSv4 environment with RPCSEC_GSS authentication and LDAP identification
- ▶ Migration from DFS to an NFSv4 environment with RPCSEC_GSS authentication and LDAP identification
- ▶ Migration from a flat-file based authentication environment to an NFSv3/v4 environment with RPCSEC_GSS authentication and LDAP identification

The AFS and DFS scenarios demonstrate the following actions:

- ▶ Migration of user information into Kerberos and LDAP
- ▶ Migration of group information into Kerberos and LDAP
- ▶ Conversion of AFS/DFS ACLs to NFSv4 ACLs
- ▶ Migration of AFS volumes and DFS filesets to an AIX 5L file system

The flat-file based (/etc/passwd and /etc/group) authentication environment scenario demonstrates the following actions:

- ▶ Migration of user information into Kerberos and LDAP
- ▶ Migration of group information into Kerberos and LDAP
- ▶ Migration from JFS/JFS2v1 file systems to JFSv2 file system
- ▶ Implementation of NFSv4 ACLs

After the migration of user and data information is complete, we present a demonstration showing how NFSv4 can be used to share data throughout your organization.

Figure 8-1 on page 191 shows the migration and implementation scenarios chosen for demonstration. The darkest arrows show the path followed by the first scenario. Scenario two takes the next path. Scenario three uses a direct approach. The dotted arrow shows an alternative path that can be used for the third scenario, but which is not demonstrated in this book.

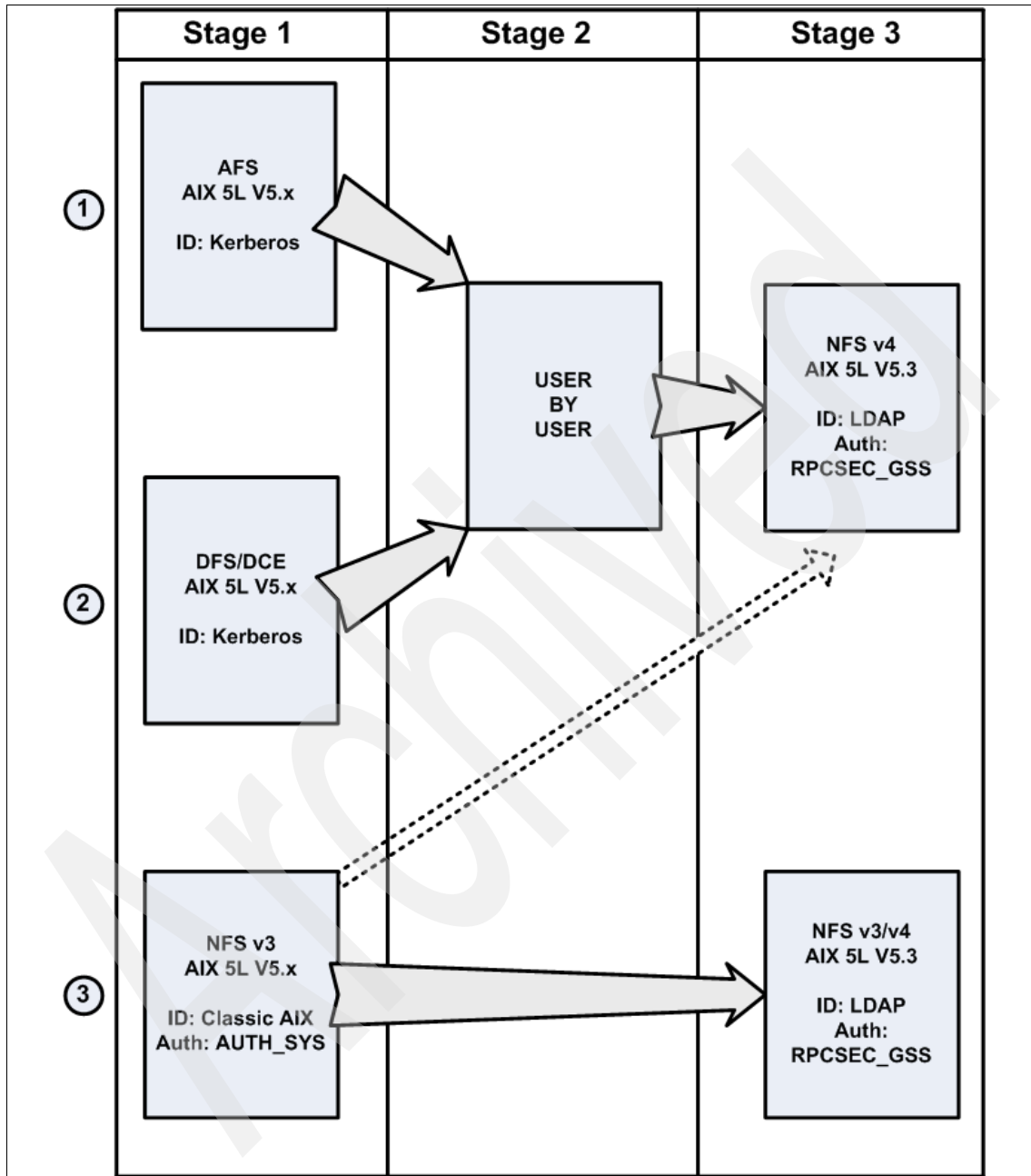


Figure 8-1 Migration scenarios covered in this book

The move from AFS and DFS to NFSv4 will be a rolling migration. We demonstrate this at a user-by-user level. The idea is to demonstrate how the two environments can coexist (it will do so during the migrations phase) with the intention of finally moving over to a homogeneous NFSv4 environment.

The third scenario demonstrates a move from a flat-file-based environment, possibly, but not necessarily, running NFSv3, to one running NFS versions 3 and 4 with LDAP identification and RPCSEC_GSS authentication. The objective is to extend on work already done in the previous IBM Redbook *Securing NFS in AIX: An Introduction to NFS V4 in AIX 5L Version 5.3*, SG24-7204, and demonstrate new features in AIX 5L V5.3 RL03 and coexistence of NFS versions 3 and 4.

The ultimate goal for anyone who wants to make full use of the capabilities provided by NFSv4 is the following environment:

- ▶ Pure NFSv4:
 - Allows a customized pseudo file system.
 - Makes all NFSv4 features accessible to all NFSv4 clients and therefore the entire environment.
- ▶ Kerberos V5 authentication
- ▶ LDAP identification



Part 4

Migrating to NFSv4

In this part, we describe the steps needed to migrate from AFS/DFS to NFSv4 and AIX 5L V5.3 RML03.

Throughout this section, we refer back to on page 194 the following figure. We attempted to develop a logical approach to the AFS/DFS migration planning process. We recommend that you use this chart as a template when carrying out your infrastructure and migration planning. We attempted to cover all possible permutations. However, it is possible that additional decision flows might be necessary, depending on the properties of specific migration environments.

For detailed descriptions and implementation considerations about the features introduced in the initial release of AIX 5L V5.3, see the IBM Redbook *Securing NFS in AIX: An Introduction to NFS V4 in AIX 5L Version 5.3*, SG24-7204. You can view or download this book at:

<http://www.redbooks.ibm.com/abstracts/sg247204.html>

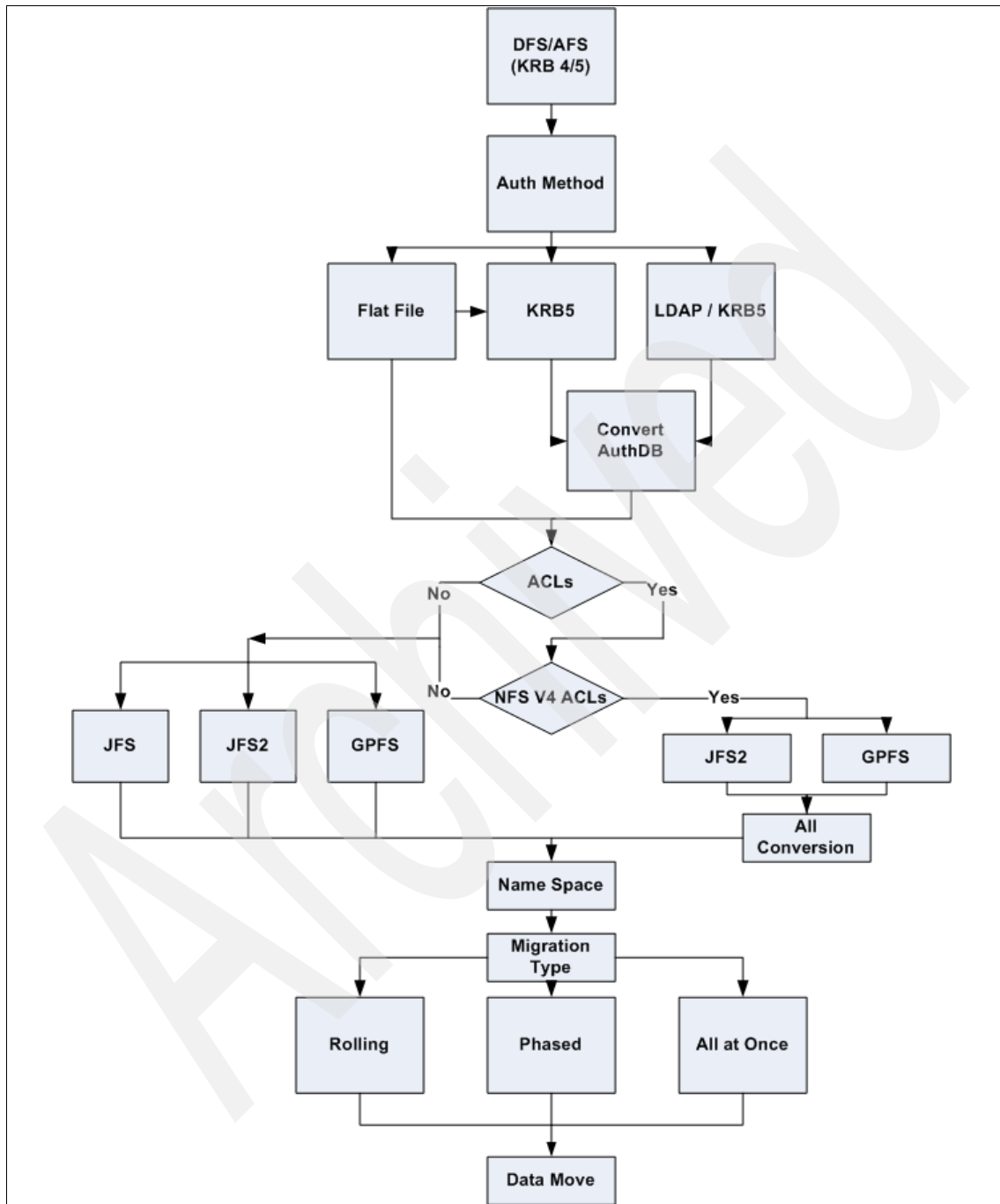


Figure P4-1: AFS/DFS migration considerations decision flowchart

NFSv3 to NFSv4 migration

This chapter provides practical information regarding the installation, deployment, and administration of NFSv4 on IBM AIX 5L V5.3. The objective is to provide several step-by-step scenarios depicting the deployment of NFSv4 across an organization. The chapter is primarily aimed at users of NFSv3; however, users new to NFS can also leverage the information to their advantage.

We discuss and demonstrate the following actions:

- ▶ General set-up
- ▶ Deploying NFSv4 in a classic NFSv3 manner, that is, without additional security
- ▶ Configuration and deployment of a pseudo root file system and global namespace and the advantages of using this mechanism
- ▶ Implementation of NFSv4 with security, using the LDAP and Kerberos environment, as described in Chapter 6, “Building an NFSv4 environment” on page 119

We discuss the following topics in this chapter:

- ▶ The test environment
- ▶ Using NFSv3 and NFSv4 side-by-side
- ▶ Migrating from NFSv3 to NFSv4
- ▶ Using NFSv3

- ▶ Using NFSv4 with NFSv3
- ▶ Adding security
- ▶ Namespace management
- ▶ Setting a different pseudo root file system

9.1 The test environment

This section describes the test environment in use when the following examples were created.

The NFSv3/v4 server is frio.itsc.austin.ibm.com.

The NFSv3/v4 (AIX 5L V5.3 RML03) client is brazos.itsc.austin.ibm.com.

The NFSv3 (AIX 5L V5.2 RML04) client is milan.itsc.austin.ibm.com.

syslogd settings

NFSv4 sends all its debug-related output to syslog. However, on AIX 5L, the syslog daemon is not enabled by default. Therefore, the following steps describe how to quickly configure syslogd:

1. Create and mount a local file system to the /var/nfs4log directory. It is generally advisable to use a separate file system for log daemons, because this ensures that the root file system is protected from filling up.
2. Add the following line to the /etc/syslog.conf file:

```
*.debug /var/nfs4log/syslog.out rotate time 1d archive \
/var/nfs4log/archive/
```
3. Restart the syslogd daemon in order to activate the changes on the running system, using the following command:

```
refresh -s syslogd
```

This method might log more information than necessary. If less detailed logging is required, debug can be changed to *error* in the /etc/syslog.conf file. This limits the amount of data written to the log file. The line now looks like:

```
*.error /var/nfs4log/syslog.out rotate time 1d archive \ /var/nfs4log/archive/
```

Again, the syslogd daemon must be restarted or refreshed after making these changes.

Tip: We recommend the use of syslogd logging during the initial implementation stage. This makes it easier to isolate problems that might occur during each step. All NFS and related daemons write to the syslog output file.

9.2 Using NFSv3 and NFSv4 side-by-side

Despite the addition of new NFSv4 functionality in AIX 5L V5.3 RML03, many customer sites require the ability to provide both NFSv3 and NFSv4 services concurrently. This can be necessary in order to support existing systems or other operating systems that presently do not offer NFSv4 implementations. Coexistence of both versions is not difficult to achieve, and a hybrid environment can be maintained indefinitely if necessary.

NFSv3 and NFSv4 co-existence

Four clients have been shaded in Figure 9-1 to provide a visual representation of version coexistence in a mixed NFS version 3 and 4 environment.

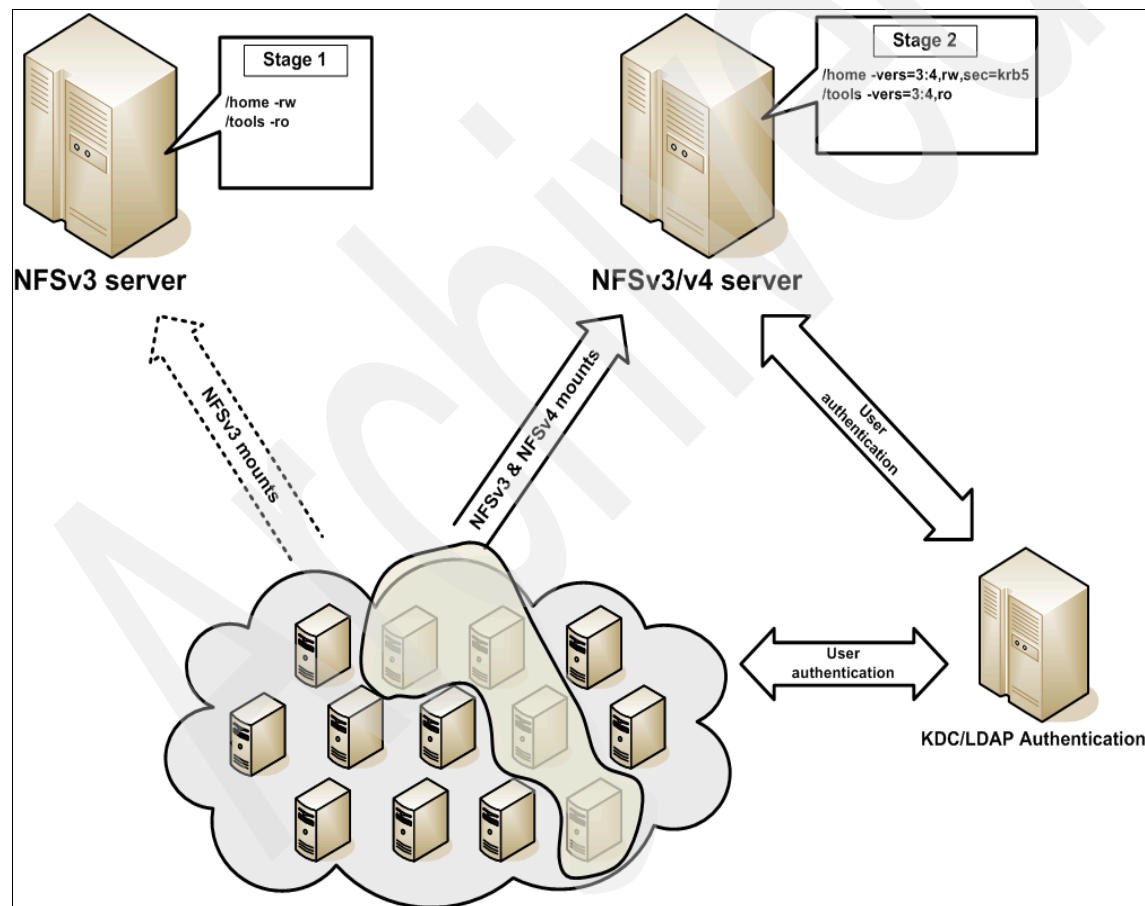


Figure 9-1 Migrating from an NFSv3 to an NFSv3/v4 environment

Two options are available when introducing NFSv4 into an existing NFSv3 environment:

- ▶ Use both protocols concurrently.
- ▶ Discard version 3 altogether in favor of version 4.

9.3 Migrating from NFSv3 to NFSv4

Figure 9-2 shows a phased migration scenario involving three stages:

- ▶ Current implementation (Stage 1)
- ▶ Intermediate implementation (Stage 2)
- ▶ Final implementation (Stage 3)

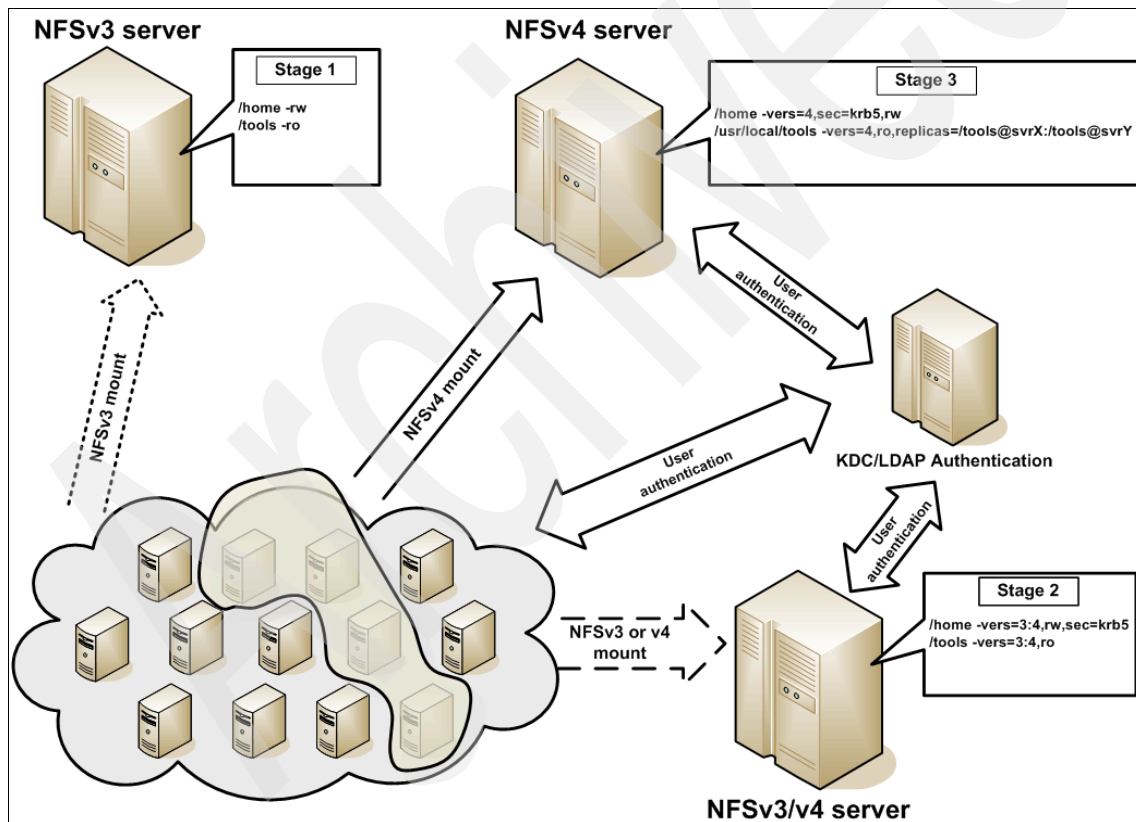


Figure 9-2 Migrating from an NFSv3 to an NFSv4 environment

Stage 2 can be bypassed if the site plans to migrate fully and immediately to NFSv4 (for example, a switch-over migration). However, this requires that all clients and servers be capable of supporting NFSv4. Therefore, the following prerequisites must be satisfied before performing the migration:

- ▶ New authentication and identification methods are ready.
- ▶ NFS servers are upgraded to AIX 5L V5.3. If another operating system is in use, it must be able to serve NFSv4 exports.
- ▶ NFS clients are upgraded to AIX 5L V5.3. If another operating system is in use, it also must be able to mount NFSv4 exports.

The scenario depicted in Figure 9-2 on page 199 is very flexible and can be used in many different combinations. This section describes each stage.

Stage 1

Because this represents the existing environment, it presumes the site is using NFSv3 in conjunction with traditional UNIX, NIS, or perhaps Kerberos V5-based authentication.

Stage 2

This can represent an intermediate or final stage, based on the desired end result. It is the final stage if the goal is to implement NFSv4 concurrently with existing version 3 services.

By this point, some clients have been migrated to NFSv4. At least one NFS server has been upgraded or replaced and is now providing NFSv3 and NFSv4 exports. This permits client access to data no matter which NFS version a specific system is using. Optionally, the exports file on the NFS server can offer the `sec=krb5` option for one or more file systems if the site's NFSv3 clients are running AIX 5L V5.3 or another version of NFS that supports the security feature.

Note: NFSv3 on AIX 5L V5.3 inherits the security features of NFSv4.

Stage 3

When this point is reached, the migration has been completed. For example purposes, and presuming the site has decided to implement the NFSv4 security features, the following changes have been made:

- ▶ Authentication and authorization environments are fully functional.
- ▶ All users are integrated into the new authentication and identification environment.
- ▶ All NFS servers are using AIX 5L V5.3 or another operating system that supports NFSv4.

- ▶ All NFS clients have been upgraded to AIX 5L V5.3 or an operating system that supports NFSv4.
- ▶ All data has been migrated from existing NFSv3 to NFSv4 style exports.

9.4 Using NFSv3

NFSv4 continues to support the existing version 3 style file system export and mount model. No changes are required to existing `/etc/exports` files to client `/etc/filesystems` files or to the automount map unless the migration plan also includes implementation of the security, replication, or referral features available with NFSv4.

Example 9-1 shows the server's view of the NFSv3 exported file systems.

Example 9-1 Exporting file systems as NFSv3

```
# cat /etc/exports
/export -rw
/export/home -rw
/export/tools -ro
/export/data -rw
#
# exportfs -va
Exported /export
Exported /export/home
Exported /export/tools
Exported /export/data
```

Example 9-2 shows the client's view of the NFSv3 exported file systems.

Example 9-2 Client view of the NFSv3 mounted file systems

```
# showmount -e friio
export list for friio:
/export      (everyone)
/export/home (everyone)
/export/tools (everyone)
/export/data (everyone)
#
# mount friio:/export /nfs
# mount friio:/export/home /nfs/home
# mount friio:/export/tools /nfs/tools
# mount friio:/export/data /nfs/data
#
# df -k
Filesystem      1024-blocks      Free %Used    Iused %Iused Mounted on
```

/dev/hd4	16384	4912	71%	1572	54%	/
/dev/hd2	1261568	185356	86%	27027	39%	/usr
/dev/hd9var	16384	8568	48%	372	16%	/var
/dev/hd3	32768	32348	2%	20	1%	/tmp
/dev/hd1	16384	15820	4%	85	3%	/home
/proc	-	-	-	-	-	/proc
/dev/hd10opt	49152	21324	57%	655	12%	/opt
/dev/fslv00	1048576	1047828	1%	109	1%	/work
/dev/fslv01	16384	16004	3%	34	1%	/home/foo
frio:/export	65536	52308	21%	1597	12%	/nfs
frio:/export/home	131072	130724	1%	4	1%	/nfs/home
frio:/export/tools	131072	130724	1%	4	1%	/nfs/tools
frio:/export/data	131072	130724	1%	4	1%	/nfs/data
#						

9.5 Using NFSv4 with NFSv3

As stated earlier, it is possible to use NFSv3 concurrently with NFSv4, and numerous reasons exist for sites to do so. For example:

- ▶ Existing NFSv3 exports must be made available to systems not yet migrated to an operating system that supports NFSv4.
- ▶ A phased migration from NFSv3 to NFSv4 is being carried out. Maintenance of existing services during the migration requires that both v3 and v4 clients be supported.

Whatever the reason, NFSv3 and NFSv4 can easily coexist. The following sections describe the steps necessary to enable AIX 5L V5.3 systems to host and mount NFSv4 exports concurrently with NFSv3 exports.

9.5.1 Configuring the NFS domain

As described in Chapter 6, “Building an NFSv4 environment” on page 119, the NFS domain name must be set before NFSv4 can be used. The current setting can be checked using the **chnfsdom** or **smitty chnfsdom** command.

If the NFS domain is not set, the output will look like the output in Example 9-3.

Example 9-3 Checking the current NFS domain

```
# chnfsdom
Current local domain: N/A
#
```

The NFS domain can be set using the **chnfsdom** command, as shown in Example 9-4.

Example 9-4 Changing the NFS domain

```
# chnfsdom itsc.austin.ibm.com
#
# chnfsdom
Current local domain: itsc.austin.ibm.com
#
```

After the NFS domain has been set, the NFS registry daemon (**nfsrgyd**) must be started. Example 9-5 shows how this is done.

Example 9-5 Starting the NFS registry daemon (nfsrgyd)

```
# startsrc -s nfsrgyd
0513-059 The nfsrgyd Subsystem has been started. Subsystem PID is 315416.
#
# lssrc -s nfsrgyd
Subsystem      Group      PID      Status
nfsrgyd        nfs        315416    active
#
```

9.5.2 Configuring the pseudo root file system

Configuration of the pseudo root file system on an NFSv4 server is very straightforward. It first requires that the pseudo root directory be available and that all exported file systems are locally mounted within this directory.

Note: The pseudo root can be changed on the server either by specifying a new value in the `/etc/exports` file or from the command line using the **chnfs** command. We recommend the latter method, using the **chnfs** command. However, changing the pseudo root will not affect file systems that have already been exported. If changes are made, they must be re-exported using the **exportfs -va** command.

By default, the root directory `/` is defined and exported as the pseudo root file system. You can check this using the **nfsd -getnodes** command. Example 9-6 on page 204 shows sample output from this command on an unconfigured system.

Example 9-6 Checking the current setting of pseudo root

```
# nfsd -getnodes
#root:public
/:/
#
```

The output in Example 9-6 indicates that the root (first /) directory is defined as /. The second / after the colon represents the public root directory. The next section describes the steps required to change the root file system to support NFSv4 pseudo root exports.

9.5.3 Exporting file systems for access to NFSv3 and NFSv4 clients

As previously discussed, NFSv3 and NFSv4 can be implemented concurrently. Example 9-7 shows file systems exported for concurrent access by both version 3 and 4 clients.

Example 9-7 Exporting file systems as both NFSv3 and v4

```
# nfsd -getnodes
#root:public
/:/
#
# exportfs -ua
#
# exportfs
exportfs: 1831-182 nothing exported
#
# cat /etc/exports/
/home -vers=3:4,rw,
/tools -vers=3:4,ro
/data -vers=3:4,rw
#
# exportfs -va
Exported /home
Exported /tools
Exported /data
#
```

9.5.4 Mounting NFSv4 exports on the clients

Client systems must be running AIX 5L V5.3 in order to mount NFSv4 exports. The **mount** command with the **-vers=4** option is used to mount an NFSv4 file system. Example 9-8 on page 205 shows the command required to mount an NFSv4 export on a client system.

Example 9-8 Mounting NFSv4 exports on the NFSv4 clients

```
# mount -o vers=4 frio:/ /nfs
#
# df -k
Filesystem      1024-blocks      Free %Used    Iused %Iused Mounted on
/dev/hd4         16384           4912   71%      1574   54% /
/dev/hd2        1261568        184140  86%     27032   39% /usr
/dev/hd9var      16384           8556   48%       372   16% /var
/dev/hd3         32768          30248   8%        24    1% /tmp
/dev/hd1         16384          15820   4%         85    3% /home
/proc            -               -       -         -     - /proc
/dev/hd10opt     49152          21324  57%       655   12% /opt
/dev/fslv00     1048576        1047828  1%        109    1% /work
/dev/fslv01      16384          16004   3%         34    1% /home/foo
frio:/           65536          52292  21%       1597   12% /nfs
#
# nfs4cl showfs

Server          Remote Path      fsid              Local Path
-----
frio.itsc.austin.ibm.com /              10:4              /nfs
#
# cd /nfs
#
# ls -la
total 9
dr-xr-xr-x  4 root    system           5 Aug 25 09:52 .
drwxr-xr-x 23 root    system          4096 Aug 24 16:14 ..
drwxr-xr-x  3 root    system           256 Aug 25 09:41 data
drwxr-xr-x  2 bin     bin             256 Aug 25 09:37 home
drwxr-xr-x  3 root    system           256 Aug 25 09:41 tools
#
```

9.5.5 Mounting NFSv3 exports on the clients

In the next example, file systems have been exported with the `-vers=3:4` option. Therefore, the exports can be mounted by both NFSv3 and version 4 clients. Example 9-9 illustrates this process.

Example 9-9 Mounting NFSv3 exports

```
# mount frio:/home /mnt
#
# mount frio:/tools /mnt2
#
# mount frio:/data /mnt3
#
```

#	df	-k					
Filesystem	1024-blocks	Free	%Used	Iused	%Iused	Mounted on	
/dev/hd4	540672	430084	21%	1753	1%	/	
/dev/hd2	2080768	925892	56%	35801	7%	/usr	
/dev/hd9var	16384	7408	55%	350	9%	/var	
/dev/hd3	3276800	3153836	4%	48	1%	/tmp	
/dev/hd1	114688	101288	12%	190	1%	/home	
/proc	-	-	-	-	-	/proc	
/dev/hd10opt	49152	21760	56%	654	6%	/opt	
/dev/lv00	540672	438436	19%	342	1%	/patch12	
/dev/lv01	5013504	4820240	4%	8678	1%	/vicepa	
AFS	72000000	72000000	0%	0	0%	/afs	
frio:/home	65536	65164	1%	3	1%	/mnt	
frio:/tools	131072	130724	1%	4	1%	/mnt2	
frio:/data	131072	130724	1%	4	1%	/mnt3	

9.5.6 Differences between NFSv3 and NFSv4 mounts

The benefit provided by NFSv4 under AIX 5L V5.3 becomes immediately apparent as soon as the difference in mount strategies is understood. On NFSv3 clients, each exported file system must be mounted individually, while only a single **mount** command is required when NFSv4 is in use. The NFSv4 server creates a pseudo view for the NFSv4 clients, so all the exported file systems are visible when a user changes directory to **/nfs**.

9.6 Adding security

Configuration of the authentication and authorization environment has already been described. This configuration can now be used to fully exploit the functionality added by NFSv4. NFSv3 has inherited this functionality on AIX 5L V5.3. The following tasks must be carried out in order to make use of the added security:

1. Configure time services.
2. Configure the LDAP/Kerberos V5 (KRB5) server.
3. Configure the LDAP/KRB5 clients.
4. Add the NFS service principal to Kerberos for all NFS servers.
5. Configure the **gssd** daemon on the NFS server.
6. Create the Kerberos V5 realm to NFS domain mapping.

For purposes of these examples, it is presumed that steps 1, 2, and 3 are already complete. See 6.1, “Environment used for demonstration scenarios” on page 120.

9.6.1 Creating NFS service principals in Kerberos

An NFS host principal must be defined for each NFS server in the environment if the security functionality is to be used. The principal definition must be in the following format:

```
nfs/<fully_qualified_name_of_NFS_server>@<Kerberos_5_realm>
```

The machine frio.itsc.austin.ibm.com will be used as the NFS server for this example. Therefore, the NFS host principal for frio is defined as follows:

```
nfs/frio.itsc.austin.ibm.com@NFS4REALM.IBM.COM
```

The steps required during creation of the NFS host principal are slightly different from those followed for the user principal; a random password is used rather than a user-defined one. Example 9-10 shows one method for creating the NFS host principal. The commands are executed on the Kerberos server, designated pecos.itsc.austin.ibm.com.

Example 9-10 Configuring an NFS host principal on a Kerberos V5 server

```
# kinit admin/admin
Password for admin/admin@NFSV4REALM.IBM.COM:
#
# kadmin.local
Attempting to bind to one or more LDAP servers. This may take a while...
kadmin.local:
kadmin.local: add_principal -e des-cbc-crc:normal -randkey
nfs/frio.itsc.austin.ibm.com
WARNING: no policy specified for
nfs/frio.itsc.austin.ibm.com@NFSV4REALM.IBM.COM;
defaulting to no policy. Note that policy may be overridden by
ACL restrictions.
Principal "nfs/frio.itsc.austin.ibm.com@NFSV4REALM.IBM.COM" created.
kadmin.local:
kadmin.local: get_principal nfs/frio.itsc.austin.ibm.com
Principal: nfs/frio.itsc.austin.ibm.com@NFSV4REALM.IBM.COM
Expiration date: [never]
Last password change: Thu Aug 25 11:13:46 CDT 2005
Password expiration date: [none]
Maximum ticket life: 1 day 00:00:00
Maximum renewable life: 7 days 00:00:00
Last modified: Thu Aug 25 11:13:47 CDT 2005 (admin/admin@NFSV4REALM.IBM.COM)
Last successful authentication: [never]
Last failed authentication: [never]
```

```
Failed password attempts: 0
Number of keys: 1
Key: vno 2, DES cbc mode with CRC-32,
no salt
```

Attributes:

```
Policy: [none]
kadmin.local: quit
#
```

9.6.2 Configuring the gssd daemon on the NFS server

To use RPCSEC_GSS, a file must be created to map between the server's keytab file and the NFS server principal. This is accomplished using the **nfshostkey** command, as shown in Example 9-11.

Example 9-11 Creating the server's hostkey map file

```
# nfshostkey -p nfs/frio.itsc.austin.ibm.com -f /etc/krb5/krb5.keytab
#
# nfshostkey -l
nfs/frio.itsc.austin.ibm.com
/etc/krb5/krb5.keytab
#
```

The **chnfs -S -B** command is executed to permit the **gssd** daemon to start automatically at boot time. Alternately, the **smitty addsecurity** command can be used. This step must be repeated on all AIX 5L NFSv4-capable servers in the environment.

Example 9-12 Starting the gssd daemon and configuring it for automatic restart

```
# chnfs -S -B
0513-059 The gssd Subsystem has been started. Subsystem PID is 295038.
#
```

9.6.3 Mapping Kerberos V5 realms to NFS domains

The **/etc/nfs/realm.map** file maps the relationship between Kerberos realms and NFS domains. It can be created using the **chnfsrtd** command. In the example environment, the Kerberos V5 realm is **NFS4REALM.IBM.COM** and the NFS domain is **itsc.austin.ibm.com**. Example 9-13 on page 209 shows the results of the **chnfsrtd** command.

Example 9-13 Creating the KRB5 realm to NFS domain mapping

```
# chnfsrtd -a NFS4REALM.IBM.COM itsc.austin.ibm.com
#
# cat /etc/nfs/realm.map
nfs4realm.ibm.com itsc.austin.ibm.com
#
# chnfsrtd
nfs4realm.ibm.com itsc.austin.ibm.com
#
```

Note: The realm entry in the `/etc/nfs/realm.map` file is not case-sensitive.

9.6.4 Creating the NFS keytab file entry

The next step is to create a keytab entry on the NFS servers. The **kadmin** command is used, as shown in Example 9-14, on the NFS server `frio`.

Example 9-14 Creating a keytab file entry on an NFS server

```
# /usr/krb5/sbin/kadmin
Authenticating as principal admin/admin@NFSV4REALM.IBM.COM with password.
Password for admin/admin@NFSV4REALM.IBM.COM:
kadmin:
kadmin: ktadd nfs/frio.itsc.austin.ibm.com
Entry for principal nfs/frio.itsc.austin.ibm.com with kvno 3, encryption type
Triple DES cbc mode with HMAC/sha1 added to keytab
WRFILE:/etc/krb5/krb5.keytab.
Entry for principal nfs/frio.itsc.austin.ibm.com with kvno 3, encryption type
ArcFour with HMAC/md5 added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal nfs/frio.itsc.austin.ibm.com with kvno 3, encryption type
AES-256 CTS mode with 96-bit SHA-1 HMAC added to keytab
WRFILE:/etc/krb5/krb5.keytab.
Entry for principal nfs/frio.itsc.austin.ibm.com with kvno 3, encryption type
DES cbc mode with RSA-MD5 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin: quit
#
```

The **ktadd** command shown in Example 9-14 creates a file called `/etc/krb5/krb5.keytab`.

At this point in the process, it is advisable to test the state of the configuration by attempting to access the NFS file system from a client system.

Answer the following questions before proceeding with the remainder of the configuration process:

- Are the keytab entries valid? Use the **ktutil** command, as shown in Example 9-15, to validate the keytab files.

Example 9-15 Checking for a valid keytab on an NFS server

```
# /usr/krb5/sbin/ktutil
ktutil:
ktutil: read_kt /etc/krb5/krb5.keytab
ktutil:
ktutil: 1
slot  KVNO  Principal
-----
      1      3 nfs/frio.itsc.austin.ibm.com@NFSV4REALM.IBM.COM
      2      3 nfs/frio.itsc.austin.ibm.com@NFSV4REALM.IBM.COM
      3      3 nfs/frio.itsc.austin.ibm.com@NFSV4REALM.IBM.COM
      4      3 nfs/frio.itsc.austin.ibm.com@NFSV4REALM.IBM.COM
ktutil: quit
#
```

- Can the server principal obtain valid tickets? For this, use the **kinit** command, as shown in Example 9-16.

Example 9-16 Verifying the NFS host principal's ability to get a valid ticket

```
# kinit -kt /etc/krb5/krb5.keytab nfs/frio.itsc.austin.ibm.com
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: nfs/frio.itsc.austin.ibm.com@NFSV4REALM.IBM.COM

Valid starting    Expires          Service principal
08/25/05 13:13:03 08/26/05 13:13:02  krbtgt/NFSV4REALM.IBM.COM@NFSV4REALM.IBM.COM
#
```

9.6.5 Configuring security on the clients.

You must follow these steps on all clients that are to mount file systems exported from the NFS server:

1. Configure time services.
2. Configure LDAP.
3. Configure Kerberos and integrated login.
4. Add the client service principal.

5. Create the client keytab file.
6. Create a map between the client keytab file and the NFS service principal.

Example 9-17 shows the execution sequence for these commands, preparing the client for the NFS environment. The commands are shown in bold print. Superscript numbering refers to explanations following the example.

Example 9-17 Configuring the NFS client for security

```
# kinit admin/admin(1)
Password for admin/admin@NFSV4REALM.IBM.COM:
#
# klist(2)
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: admin/admin@NFSV4REALM.IBM.COM

Valid starting    Expires          Service principal
08/25/05 14:19:00 08/26/05 14:18:57
krbtgt/NFSV4REALM.IBM.COM@NFSV4REALM.IBM.COM
#
# /usr/krb5/sbin/kadmin(3)
Authenticating as principal admin/admin@NFSV4REALM.IBM.COM with password.
Password for admin/admin@NFSV4REALM.IBM.COM:
kadmin:
kadmin: add_principal -e des-cbc-crc:normal -randkey
nfs/brazos.itsc.austin.ibm.com(4)
WARNING: no policy specified for
nfs/brazos.itsc.austin.ibm.com@NFSV4REALM.IBM.COM;
        defaulting to no policy. Note that policy may be overridden by
        ACL restrictions.
Principal "nfs/brazos.itsc.austin.ibm.com@NFSV4REALM.IBM.COM" created.
kadmin:
kadmin: ktadd nfs/brazos.itsc.austin.ibm.com(5)
Entry for principal nfs/brazos.itsc.austin.ibm.com with kvno 3, encryption type
Triple DES cbc mode with HMAC/sha1 added to keytab
WRFILe:/etc/krb5/krb5.keytab.
Entry for principal nfs/brazos.itsc.austin.ibm.com with kvno 3, encryption type
ArcFour with HMAC/md5 added to keytab WRFILe:/etc/krb5/krb5.keytab.
Entry for principal nfs/brazos.itsc.austin.ibm.com with kvno 3, encryption type
AES-256 CTS mode with 96-bit SHA-1 HMAC added to keytab
WRFILe:/etc/krb5/krb5.keytab.
Entry for principal nfs/brazos.itsc.austin.ibm.com with kvno 3, encryption type
DES cbc mode with RSA-MD5 added to keytab WRFILe:/etc/krb5/krb5.keytab.
kadmin:
kadmin: quit
# kinit -kt /etc/krb5/krb5.keytab nfs/brazos.itsc.austin.ibm.com(6)
#
# klist(7)
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
```

Default principal: nfs/brazos.itsc.austin.ibm.com@NFSV4REALM.IBM.COM

```
Valid starting    Expires          Service principal
08/25/05 14:22:26 08/26/05 14:22:26
krbtgt/NFSV4REALM.IBM.COM@NFSV4REALM.IBM.COM
#
# nfshostkey -p nfs/brazos.itsc.austin.ibm.com -f /etc/krb5/krb5.keytab(8)
#
# nfshostkey -l(9)
nfs/brazos.itsc.austin.ibm.com
/etc/krb5/krb5.keytab
#
# kdestroy(10)
#
# kinit -kt `tail -n 1 /etc/nfs/hostkey` `head -n 1 /etc/nfs/hostkey`(10)
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: nfs/brazos.itsc.austin.ibm.com@NFSV4REALM.IBM.COM

Valid starting    Expires          Service principal
08/25/05 14:38:00 08/26/05 14:37:59
krbtgt/NFSV4REALM.IBM.COM@NFSV4REALM.IBM.COM
#
# chnfs -S -B(11)
0513-059 The gssd Subsystem has been started. Subsystem PID is 352346.
#
```

The following list explains the example commands in Example 9-17 on page 211:

1. The following command obtains a Kerberos ticket-granting ticket (TGT) for the admin principal:
`kinit admin/admin`
2. The following command verifies the ticket obtained in step 1:
`klist`
3. The following command starts the Kerberos administrative interface, permitting the authenticated admin/admin principal to execute account management tasks:
`/usr/krb5/sbin/kadmin`
4. The following command adds the nfs principal for host brazos to the Kerberos realm's database:
`add_principal -e des-cbc-crc:normal -randkey \
nfs/brazos.itsc.austin.ibm.com`

5. The following command adds an entry for the nfs principal to the `/etc/krb5/krb5.keytab` file:

```
ktadd nfs/brazos.itsc.austin.ibm.com
```
6. The following command obtains the key for the nfs principal from the keytab file `/etc/krb5/krb5.keytab`:

```
kinit -kt /etc/krb5/krb5.keytab nfs/brazos.itsc.austin.ibm.com
```
7. Same as step 2.
8. The following command configures the nfs host key for brazos:

```
nfshostkey -p nfs/brazos.itsc.austin.ibm.com -f /etc/krb5/krb5.keytab
```
9. The following command verifies the host key:

```
nfshostkey -l
```
10. The following commands destroy the session's existing ticket information, and then obtain a new ticket using the key located in the `/etc/nfs/hostkey` file. This is done to verify the correctness of the preceding steps.

```
kdestroy  
kinit -kt `tail -n 1 /etc/nfs/hostkey` `head -n 1 /etc/nfs/hostkey`
```
11. The following command starts the `gssd` daemon and sets it to start automatically during future reboots:

```
chnfs -S -B
```

The client configuration is now complete.

9.6.6 Exporting NFS file systems with security

Next, we must modify entries in the `/etc/exports` file on the NFS server. Example 9-18 shows the addition of the `-sec=krb5` option to the existing entries, and subsequent re-exporting of these file systems to enable the new option.

Example 9-18 Adding Kerberos V5 to the exports on the NFS server

```
# cat /etc/exports
/home -vers=3:4,sec=krb5,rw
/tools -vers=3:4,sec=krb5,ro
/data -vers=3:4,sec=krb5,rw
#
# exportfs -va
exportfs: 1831-187 re-exported /home
exportfs: 1831-187 re-exported /tools
exportfs: 1831-187 re-exported /data
#
```

The file systems are now ready to be mounted by the client systems.

9.6.7 Mounting an NFSv4 exported file system

Example 9-19 demonstrates the procedure for mounting an NFSv4 export on an AIX 5L V5.3 client.

Example 9-19 Mounting NFSv4 exports on AIX 5L V5.3 client with Kerberos V5 enabled

```
# mount -o vers=4,sec=krb5 frio:/ /nfs
#
# nfs4cl showfs /nfs
```

Server	Remote Path	fsid	Local Path
frio.itsc.austin.ibm.com	/	10:4	/nfs

Current Server: frio.itsc.austin.ibm.com:nfs

options :
rw,intr,rsz=32768,wsz=32768,timeo=100,retrans=5,maxgroups=0,acregmin=3,acregmax=60,acdirmin=30,acdirmax=60,minpout=1250,maxpout=2500,sec=sys:krb5:krb5i:krb5p

```
#
# mount
```

node	mounted	mounted over	vfs	date	options
	/dev/hd4	/	jfs2	Aug 10 10:19	rw,log=/dev/hd8
	/dev/hd2	/usr	jfs2	Aug 10 10:19	rw,log=/dev/hd8
	/dev/hd9var	/var	jfs2	Aug 10 10:19	rw,log=/dev/hd8
	/dev/hd3	/tmp	jfs2	Aug 10 10:19	rw,log=/dev/hd8
	/dev/hd1	/home	jfs2	Aug 10 10:20	rw,log=/dev/hd8
	/proc	/proc	procfs	Aug 10 10:20	rw
	/dev/hd10opt	/opt	jfs2	Aug 10 10:20	rw,log=/dev/hd8
	/dev/fs1v00	/work	jfs2	Aug 15 11:34	rw,log=/dev/hd8
frio	/	/nfs	nfs4	Aug 25 19:13	vers=4,sec=krb5

```
#
```

As shown in Example 9-19, the client has successfully mounted the NFSv4 file systems exported by the NFSv4 server frio.

9.7 Namespace management

The *single namespace*, introduced in NFSv4, is a new concept for AIX 5L. The ramifications of this change, as well as the benefits it provides, must be understood clearly before implementing it in a production environment.

From an administrative point of view, a single namespace reduces the number of mounts a client makes per server to one. With previous versions of NFS, each client was required to mount individual NFS exports provided by a given server. A single, or unified, namespace enables the server to render a single pseudo file system view to the client; therefore, only one **mount** command is required for a client to obtain access to the namespace. Each site must consider its current pattern of NFS exports and per-client access requirements in order to gain the optimal benefits afforded by this feature.

An example of the difference between a single-mount NFSv4 strategy and the method available with version 3 can best be expressed by comparing the processing required for each. Example 9-20 shows the `/etc/exports` file required to export five file systems from an NFS server.

Example 9-20 The `/etc/exports` file with NFSv3

```
/home -rw
/usr/local -ro
/var/db2/v81/DB -rw
/temp/scratch -rw
/www -ro
```

The NFSv3 client must then mount each exported file system individually, as shown in Example 9-21.

Example 9-21 NFSv3 client mounting five file systems one at a time

```
# mount -v nfs -o rw serverY:/home /mount1
#
# mount -v nfs -o ro serverY:/usr/local /mount2
#
# mount -v nfs -o rw serverY:/var/db2/v81/DB /mount3
#
# mount -v nfs -o rw serverY:/exports/scratch /mount4
#
# mount -v nfs -o ro serverY:/www /mount5
#
```

The NFSv3 server is exporting five file systems. This requires the client to perform individual **mount** commands, producing five different mount points. The processing increases significantly if you have 30 clients performing the individual mount. If the decision is then made to move `/temp/scratch` to `/scratch`, all 30 clients must first unmount this file system, and then it must be unexported on the NFS server. After the file system has been modified, the server's `/etc/exports` file must be altered to reflect the change and then re-exported. The altered file system must next be mounted again on the client systems. If the client systems

are configured to mount NFS file systems at boot time, each machine's `/etc/filesystems` file must be edited to reflect the changed mount point.

It quickly becomes apparent that NFSv3 is not easily scalable and might be difficult to manage in a complex environment.

9.7.1 How does the NFSv4 namespace help?

Using the same directory structure discussed in Example 9-20 on page 215, we now discuss the differences inherent in the NFSv4 implementation.

As previously discussed, NFSv3 remains the default version in AIX 5L V5.3. If no version is specified when exporting or mounting a file system, AIX 5L assumes the use of version 3. To make use of NFSv4, file systems must be exported and mounted using the **vers=4** option, as shown in Example 9-22.

Example 9-22 Making use of NFSv4

```
/home -vers=4,rw
/usr/local -vers=4,ro
/var/db2/v81/DB -vers=4
/temp/scratch -vers=4,rw
/www -vers=4,ro
```

The client then issues the following command to mount the NFSv4 exports:

```
mount -o vers=4 serverY:/ /nfs
```

This **mount** command tells serverY that the client is requesting a view of its unified namespace. After the server grants access to the view, the client mounts it locally to the `/nfs` mount point.

Example 9-23 shows the `ls -la /nfs` output on the client after the NFSv4 exports have been mounted.

Example 9-23 Client view of the NFSv4 mount

```
# ls -la
total 12
dr-xr-xr-x  6 root    system      7 Aug 30 11:05 .
drwxr-xr-x 22 root    system    4096 Aug 30 09:58 ..
drwxr-xr-x  4 bin      bin       256 Aug 19 13:49 home
dr-xr-xr-x  2 root    system      3 Aug 30 11:05 temp
dr-xr-xr-x  2 root    system      3 Aug 30 11:05 usr
dr-xr-xr-x  2 root    system      3 Aug 30 11:05 var
drwxr-xr-x  2 root    system    256 Aug 30 09:54 www
#
```

The implementation and management differences become significant when the NFSv4 and NFSv3 procedures are compared.

Under NFSv3, the following steps are required:

1. The `/etc/exports` file on the server is created.
2. The clients explicitly mount each file system exported by the server.

Changes to existing NFSv3 exports require the following task list:

1. Unmount the file system on all clients.
2. Unexport the file system on the server.
3. Make the necessary changes to the file system.
4. Export the changed file system.
5. Remount the changed file system on all clients.

If NFSv4 is used, the following steps are required:

1. The `/etc/exports` file on the server is created.
2. The clients perform one mount per NFSv4 server, thus obtaining access to all the server's exported file systems under a single, unified tree.

Changes to NFSv4 exports only require actions on the server side. The client's view is dynamically updated to reflect the changes on the server. For example, the following steps are required to remove `/home` from a server's existing exports:

1. Run:

```
exportfs -u /home
```
2. Edit the `/etc/exports` file and remove the appropriate entry.

This description showcases the ease of client and server management provided by NFSv4 and its unified namespace.

9.7.2 Enhancing classic NFSv4 exports using the `exname` option

The examples described earlier can be further enhanced by incorporating the AIX-specific *alias tree model* known as **exname**. In previous sections, we discussed the use of standard NFSv3 exports and the pseudo root feature to simplify client management. The alias tree model adds even more flexibility to NFSv4 exports. For example, consider a requirement to export the `/var/db2/v81/DB` file system. If this is exported using the classic model, the full path is visible under the NFS root directory. The **exname** feature is extremely useful if, for example, users require access only to the DB subdirectory or a requirement exists to conceal the `/var` and `/var/db2` directories when this file

system is exported to clients. The **exname** feature allows file systems to be exported as individual subdirectories of the `nfsroot`. Therefore, the above directory can be exported as `/nfs/DB`.

Example 9-24 shows a namespace built using the **exname** option.

*Example 9-24 Using the **exname** option to further enhance the namespace*

```
/exports -nfsroot
/home -vers=4,rw,exname=/exports/home
/usr/local -vers=4,ro,exname=/exports/local
/var/db2/v81/DB -vers=4,rw,exname=/exports/DB
/temp/scratch -vers=4,rw,exname=/exports/scratch
/www -vers=4,ro,exname=/exports/www
```

The contents of this file are as follows:

1. The pseudo root of the NFSv4 server is set using the `/exports -nfsroot` line.
2. File systems are exported in the same manner as the classic NFSv4 model, but with the addition of the **exname** option. For this example, all file systems exported by the NFS server are visible under the `/exports` mount point on the client side.

Note that **exname** option does not display the full path of the exported directories. This hides the actual location of the file system, while simultaneously minimizing the number of directories that must be traversed by client systems.

The **mount** command on the client remains the same as that used under the classic NFSv4 export model.

Example 9-25 shows the results of an `ls -la /nfs` command executed on a client machine.

*Example 9-25 Client view of the **exname** exported file systems*

```
# ls -al /nfs
total 10
dr-xr-xr-x  6 root    system      7 Aug 30 09:56 .
drwxr-xr-x 22 root    system    4096 Aug 30 09:58 ..
dr-xr-xr-x  2 root    system        3 Aug 30 09:56 DB
drwxr-xr-x  4 bin     bin       256 Aug 19 13:49 home
drwxr-xr-x  3 root    system    256 Aug 15 17:03 local
drwxr-xr-x  2 root    system    256 Aug 30 09:55 scratch
drwxr-xr-x  2 root    system    256 Aug 30 09:54 www
#
```

Extending a previous example, moving an exported `/temp/scratch` to `/scratch` in an NFSv4 setting involves only the following steps on the server side. No

changes are required from the client's perspective, because the server will propagate the change automatically.

1. Unexport the /temp/scratch directory:

```
exportfs -u /temp/scratch
```

2. Move /temp/scratch to /scratch.

3. Update /etc/exports with the new location of /scratch.

4. Run the following **exportfs** command:

```
exportfs -va
```

The amount of downtime can further be reduced on the /temp/scratch file system by keeping it exported and copying it to /scratch rather than using the **mv** command. After the copy is complete, the /etc/exports file is updated and the following command sequence is executed:

```
exportfs -u /temp/scratch
exportfs -va
```

Note: No modifications are required on the client systems. The client view is updated dynamically when /temp/scratch is unexported.

Example 9-26 shows the clients' view when /temp/scratch is unexported.

Example 9-26 Client view when /temp/scratch is unexported on the server

```
# ls -al /nfs
total 10
dr-xr-xr-x  5 root    system      7 Aug 30 10:07 .
drwxr-xr-x 22 root    system     4096 Aug 30 09:58 ..
dr-xr-xr-x  2 root    system        3 Aug 30 09:56 DB
drwxr-xr-x  4 bin     bin       256 Aug 19 13:49 home
drwxr-xr-x  3 root    system     256 Aug 15 17:03 local
drwxr-xr-x  2 root    system     256 Aug 30 09:54 www
#
```

Example 9-27 shows the client view after the new /scratch file system has been exported.

Example 9-27 Client view when /scratch is exported on the server

```
# ls -la
total 10
dr-xr-xr-x  6 root    system      7 Aug 30 10:08 .
drwxr-xr-x 22 root    system     4096 Aug 30 09:58 ..
dr-xr-xr-x  2 root    system        3 Aug 30 09:56 DB
drwxr-xr-x  4 bin     bin       256 Aug 19 13:49 home
```

```

drwxr-xr-x  3 root    system      256 Aug 15 17:03 local
drwxr-xr-x  2 root    system      256 Aug 30 10:07 scratch
drwxr-xr-x  2 root    system      256 Aug 30 09:54 www
#

```

Figure 9-3 depicts the view the NFSv4 server builds for the clients.

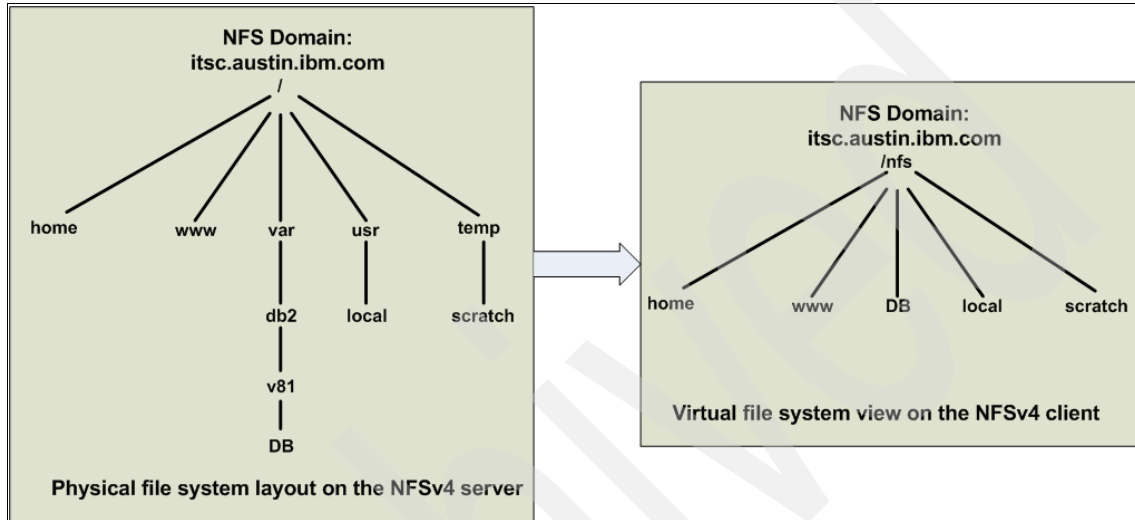


Figure 9-3 The virtual NFSv4 exported file system view

9.8 Setting a different pseudo root file system

You can use the `nfsd` command to change the pseudo root file system.

Perform the following steps on the server to set the pseudo root to `/exports` and export it to the clients:

1. Check the current settings for pseudo root by running the `nfsd -getnodes` command.
2. Check that no NFS file systems are exported by using the `exportfs` command.
3. If there are exported file systems, run the `exportfs -ua` command to unexport them.
4. Stop all NFS server processes by running `/etc/nfs.clean`.

5. Change the pseudo root with **chnfs -r /exports** or **smitty chrootfh**.

The **chnfs** command:

- a. Changes the entries within the AIX Subsystem Resource for **nfsd**. This can be verified using the **lssrc -Ss nfsd** command.
- b. Starts the **nfsd** in command line mode outside the control of the Subsystem Resource Controller.

Two options are available to complete the transition:

- Reboot the system so that all changes take effect.
- Stop all NFS processes using the **/etc/nfs.clean** command and restart NFS using the **/etc/rc.nfs** command.

Planning a migration from DFS

This chapter discusses special issues that must be considered when migrating from a DCE/DFS environment to NFSv4. We discuss the following topics in this chapter:

- ▶ An overview of DCE/DFS services and architecture
- ▶ Security (authentication)
- ▶ File ACLs
- ▶ Component-specific migration considerations
- ▶ ACL migration considerations
- ▶ Existing namespace and data migration

10.1 An overview of DCE/DFS

DCE and DFS are discrete products. Distributed Computing Environment (DCE) is considered a middleware product and is an architecture on which other applications can be built. Distributed File Service (DFS) is a DCE-based application; it cannot be installed or used in the absence of underlying DCE services. DCE can stand alone, without DFS.

As noted earlier, DCE provides security and authentication services to the DFS application. It also provides the base remote procedure call (RPC) environment, time synchronization, directory services, auditing, and other utilities that can be used to build applications such as transaction processing monitors, Object Request Broker (ORB) services, backup and restore services, and printing systems.

10.1.1 Servers and clients

In DCE, a server provides a service, while a client consumes or uses a service. Given TCP/IP, a service is offered through an *endpoint*, which is a combination of an IP address and port number where such a service listens for inbound connections. Example 10-1 shows an example of an endpoint (IP addresses have been replaced with “x”).

Example 10-1 Example of an endpoint

```
dcecp> rpcentry show /./milan.itsc.austin.ibm.com_ch
{257df1c9-c6d3-11ca-8554-08002b1c8f1f 1.0
  {ncadg_ip_udp 9.x.x.xx}
  {ncacn_ip_tcp 9.x.x.xx}}
{b238d29a-0ab2-11da-91dd-000629b91ea4}
dcecp>
dcecp> endpoint show -interface {257df1c9-c6d3-11ca-8554-08002b1c8f1f 1.0}
{{object b238d29a-0ab2-11da-91dd-000629b91ea4}
 {interface {257df1c9-c6d3-11ca-8554-08002b1c8f1f 1.0}}
 {binding {ncacn_ip_tcp 9.x.x.xx 35230}}
 {annotation {CDS Server clerkserver:
 /.../itsc.austin.ibm.com/milan.itsc.aus}}}}

{{object b238d29a-0ab2-11da-91dd-000629b91ea4}
 {interface {257df1c9-c6d3-11ca-8554-08002b1c8f1f 1.0}}
 {binding {ncadg_ip_udp 9.x.x.xx 36660}}
 {annotation {CDS Server clerkserver: /.../itsc.austin.ibm.com/milan.itsc.aus}}}}
```

In Example 10-1, the CDS clearinghouse on milan.itsc.austin.ibm.com is listening on TCP (ncacn_ip_tcp) port 35230, while a UDP (ncadg_ip_udp) interface is active on port 36660.

When a client requires a connection to a server, it queries CDS to learn from which IP address the machine has registered and then contacts the dced process running at that IP address to request the endpoint on which a requested server interface is listening. It then attempts to contact that endpoint. Note that a server machine is also a client; a CDS server also runs a cdsclerk client process that makes requests to the server process. Other machines that offer no services of their own are considered clients.

Note: The client/server distinction also applies to non-DCE services. A transaction processing server process can run on a machine where only DCE client services are offered, for example, or on a machine offering CDS or security services.

DCE provides for replication of all its core services. A site can choose to run one or more security, CDS, or time servers for example. In all cases, one acts as the master, while all others in the same environment are referred to as slaves or replica servers. Replicas provide read-only copies of the security registry, namespace, or other services, and sites can opt to configure certain groups of clients to *prefer* a specific server based on load, capacity, geographic location, or other criteria as desired. If a server fails to respond within a given time frame, the client fails over automatically to the next server on its preference list. If no such list exists, the client can query the CDS server to determine whether another server is registered under the same interface and version. Exhausting all available servers registered in CDS under a specific interface causes the client to return an error.

10.1.2 Cells

The DCE environment involves the creation of a management domain known as a cell, which is composed of all the client and server systems sharing a given set of directory, time, and security services. A cell's name usually reflects the DNS domain under which it operates (for example, ibm.com@), but this is not mandatory. The **getcellname** command, or the special variable **_c**, can be used in a **dcecp** command to view the name of an existing cell, as shown in Example 10-2.

Example 10-2 Using dcecp to view the cell name

```
dcecp> echo $_c
/.../dce_test.itsc.austin.ibm.com
dcecp>
dcecp> getcellname
/.../dce_test.itsc.austin.ibm.com
dcecp>
```

Another **dcecp** command can be used to view all the machines in a cell, as shown in Example 10-3.

Example 10-3 Using dcecp to view server names

```
> dcecp -c cell show
{secservers
  /.../dce_test.itsc.austin.ibm.com/subsys/dce/sec/server1
  /.../dce_test.itsc.austin.ibm.com/subsys/dce/sec/server2}
{cdsservers
  /.../dce_test.itsc.austin.ibm.com/hosts/server1
  /.../dce_test.itsc.austin.ibm.com/hosts/server2}
{dtsservers
  /.../dce_test.itsc.austin.ibm.com/hosts/server1.itsc.austin.ibm.com/dts-entity}
{hosts
  /.../dce_test.itsc.austin.ibm.com/hosts/corsair.itsc.austin.ibm.com
  /.../dce_test.itsc.austin.ibm.com/hosts/server1
  /.../dce_test.itsc.austin.ibm.com/hosts/server2
  /.../dce_test.itsc.austin.ibm.com/hosts/server2.itsc.austin.ibm.com
  /.../dce_test.itsc.austin.ibm.com/hosts/test0.itsc.austin.ibm.com
  /.../dce_test.itsc.austin.ibm.com/hosts/test1.itsc.austin.ibm.com}
}
```

A large production environment might involve thousands of client machines and dozens of servers, depending on the extent of the services offered and the load placed on services such as CDS and security.

10.1.3 Cross-cell communications

A relationship known as an *inter-cell trust* can be implemented in order to exchange information among multiple DCE cells where such a requirement exists. This involves creating an account, usually named something like `krbtgt/othercell.cell.com`, and the execution of the **dcecp registry connect** command on security servers located in both cells.

The presence of such trust relationships must be addressed during a migration to a new enterprise file system architecture such as NFSv4, because users might need to retain access to resources located in a foreign cell.

Important: Assessing the impact of changes to existing cross-cell relationships is an essential component of the migration planning process.

10.1.4 Caching

As with AFS, DFS makes use of client-side caching to improve performance and decrease the load on servers. The characteristics of the cache on a given system

are tunable using the **cm** family of commands. Example 10-4, shows the cache usage on a client machine. Other commands permit operations such as modification of the cache size, timeout values before the cache is flushed, and the manual flushing of the cache.

Example 10-4 Checking the cache usage on a client machine

```
> cm getcachesize  
DFS is using 64236 of the cache's available 69632 1K byte (disk) blocks.  
>
```

10.1.5 Aggregates and filesets

An aggregate is a DFS name for a UNIX logical volume or disk partition that has been allocated for use by DFS. Two types of aggregates can be used by DFS: raw and standard UNIX (for example, UNIX File System, UFS, or JFS).

Raw aggregates are created using the LFS or Episode file system. They are able to make use of the DFS replication abilities (as described in the following section). Multiple filesets can be created within a single raw aggregate, resized as needed using the **fts** suite of commands, and moved from one server to another as load and disk space dictate.

UNIX aggregates cannot make use of the DFS replication or fileset manipulation capabilities. They are visible within the federated namespace offered by DFS, but are managed in the same manner as traditional file systems. Only one fileset can be created on a UNIX aggregate, unlike the one-to-many mapping of aggregates and filesets under the Episode model.

10.1.6 Replication

DFS filesets, if based on the Episode file system, can be configured for either *release* or *scheduled* replication. The former requires manual user intervention using the **fts release** command in order to replicate a read/write fileset to one or more read-only copies. Scheduled replication establishes a fixed interval when automatic replication will occur from a read/write copy to its read-only copies.

Read-only replicas can be located on the same or different aggregates and on the same or a different server. The former strategy produces basic data security through the presence of multiple copies, while the latter adds resilience based on the geographic distance between copies.

10.2 Component-specific migration considerations

In this section, we discuss component-specific migration considerations.

10.2.1 Authentication services

The migration of existing authentication services from a DCE/DFS environment requires a great deal of planning, because the existing architecture might make use of more than one security data source. Many sites use the AIX 5L integrated login or the Sun Solaris Pluggable Authentication Model (PAM) to manage concurrent access to accounts located within a DCE security registry, Microsoft Active Directory server, MIT Kerberos V5, or IBM Network Authentication Services along with traditional flat files. Migrating to an NFSv4 environment might require that DCE/DFS and other accounts be loaded into an MIT Kerberos or IBM Network Authentication Services database in order to make use of, for example, the enhanced ACL services and improved security of NFSv4. Figure 10-1 shows an example of concurrent authentication services.

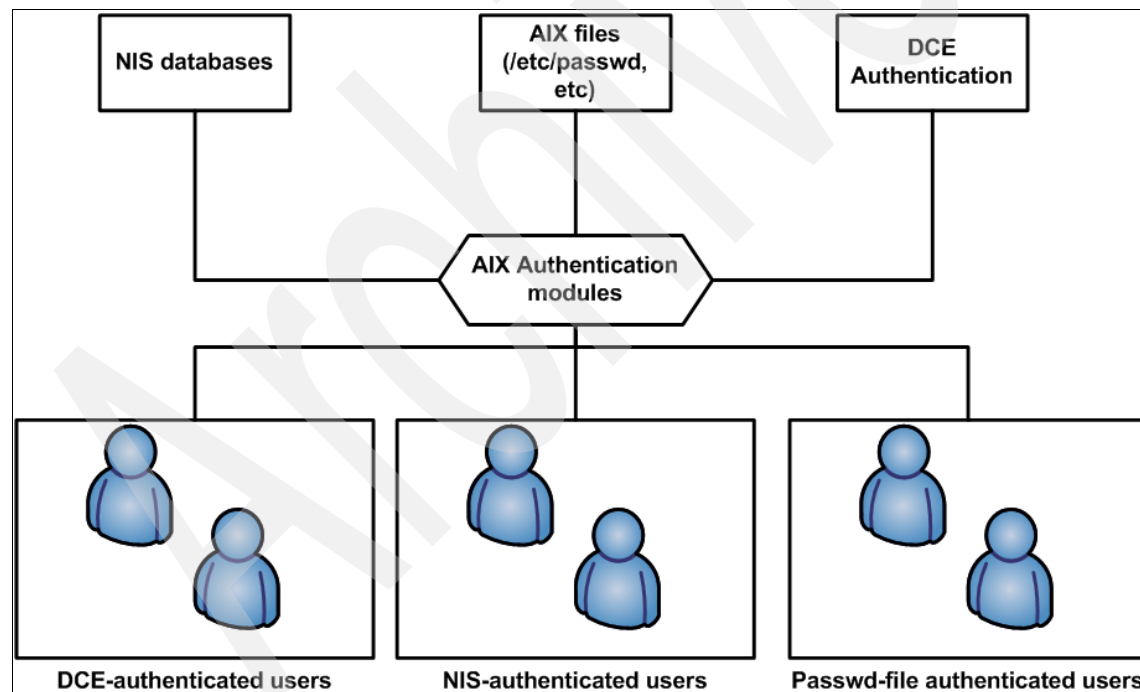


Figure 10-1 An example of concurrent authentication services

10.2.2 DCE/DFS principal and group considerations

The DCE/DFS environment places few practical limitations on the length of principal or group names, or on the number of groups to which a principal can be a member. The AIX 5L environment, however, maintains a strict 8-character limit on user ID and group names. This can make it difficult, if not impossible, to migrate certain DCE user accounts into a new environment involving traditional passwd files. Migration to a Kerberos, Active Directory, or IBM Network Authentication Services environment should not present this problem because none of these products observe the 8-character limitation.

Note: The 8-character restriction has been lifted starting in AIX 5L Version 5.3 through use of the `max_logname` system variable, which must be changed using the `chdev` command. *AIX 5L Differences Guide Version 5.3 Edition*, SG24-7463, fully documents this change.

Another issue involves the case-sensitivity of user principal names. While DCE, MIT Kerberos, IBM Network Authentication Services, and Active Directory are case-sensitive, LDAP is not. The login ID JSMITH is identical to jsmith in an LDAP context; if both accounts exist in the DCE registry, only one will be migrated to a new LDAP environment. Sites planning a migration from DCE to an LDAP-based authentication system should closely examine current principal listings in order to determine whether existing account naming practices will cause migration failures for some accounts. The `dcecp` command **principal catalog -simplename** will generate a list of existing users in order to facilitate the review process. For ease of use, the following `dcecp` command creates a text file containing a list of all users:

```
dcecp -c principal catalog -simplename > users.txt
```

Similarly, while DCE does not limit group membership on a per-user basis, AIX 5L and other operating systems frequently limit the number of groups to which a given user can belong. The AIX 5L group limit as of Version 5.3 is 128 groups per user; therefore, it might be necessary to evaluate current group membership practices within DCE/DFS and possibly restructure them to accommodate an NFSv4 environment. To test the group membership for a given DCE user, simply run the `dcecp` command and issue the command `user show <user_id>`. You will see a line similar to the following line in the result:

```
{groups subsys/dce/dfs-admin dce-ibm dce-all build testteam change-team  
system-admin}
```

In this example, the user is a member of seven groups.

In addition, you can use the **dcecp** command in the following way to see the complete list of an existing group's members:

```
dcecp -c group list <group_name> -simplename
```

10.2.3 Migrating accounts from DCE to Kerberos V5

The preferred method for migrating an existing DCE security environment to an IBM Network Authentication Services realm involves use of the tools documented in the *IBM Distributed Computing Environment Version 3.2 for AIX and Solaris: DCE Security Registry and LDAP Integration Guide*. The exact steps are beyond the scope of this document, but are presented in greater detail in the IBM Redbook *DCE Replacement Strategies*, SG24-6935. Specific considerations include:

- ▶ All existing security servers must be running DCE V3.2.
- ▶ A partial migration, that is, one involving only a subset of existing DCE principals, is not supported. Only the full security registry can be migrated using this process.
- ▶ After a *migration slave* has been created, updates made to its LDAP database are not propagated back to existing security servers. Therefore, it is possible for the existing and LDAP services to provide inconsistent data. The preferred method involves:
 - Creating a migration slave; this initially populates the LDAP environment with data from the existing environment.
 - Migrating the remaining existing security replicas to LDAP security slave status.
 - Unconfiguring the existing master after the stability of the new environment has been satisfactorily demonstrated.

The **mkseckrb5** utility

The **mkseckrb5** utility is delivered as part of AIX 5L and can be used to migrate existing local user accounts from `/etc/passwd` into a Kerberos V5 realm. However, it cannot be used when migrating DCE users because it is not aware of accounts stored in the DCE registry. This command can be of some use during a DCE migration if a subset of users maintain accounts outside the DCE environment, but the caveat described earlier regarding duplicate accounts must be observed. If user `jsmith` has both a DCE and an AIX 5L account, only one can be migrated to an MIT Kerberos-based or LDAP-based account management subsystem. Merging the user's data into a single, unified namespace might provide a solution to this type of problem.

Custom account migration solutions

The method used to migrate existing user accounts from a DCE registry to LDAP or Kerberos V5 might be limited by the type of migration, such as user-by-user, chosen for a given environment. Therefore, it might be necessary to create local scripts to retrieve fields, such as principal name, home directory, GID, and UID, from the registry. There are several methods to accomplish this:

- ▶ The unsupported **sec_salvage_db** utility can be used to create a flat ASCII copy of the DCE registry, after which a parser can be written to read in this data using a language such as Perl, TCL, or the UNIX shell.
- ▶ The output from individual **dcecp -c user show** commands can be read in a similar fashion and parsed by another script.
- ▶ A program can be written in C that calls DCE functions to retrieve user account information from the registry. The resulting data can then be inserted into a Kerberos V5 realm's database using that product's family of account management functions.

The code fragment in Example 10-5 provides details of the functions required to extract user data from the DCE registry.

Example 10-5 Code sample to extract user data from the DCE registry

```
sec_id_parse_name (rgy_site, input_name, cell_name, NULL, prin_name, NULL,
                  &status);
ERR_CHECK ("sec_id_parse_name", status, error_status_ok, TRUE);

printf (" Principal name: '%s'\n", (char *)prin_name);
printf (" Cell name: '%s'\n", (char *)cell_name);

printf ("\nGetting principal info...\n");

sec_rgy_cursor_reset(&cursor_p);
ERR_CHECK ("sec_rgy_cursor_reset (p)", status, error_status_ok, TRUE);

sec_rgy_pgo_get_by_name (rgy_site, sec_rgy_domain_person, prin_name,
                        &cursor_p, &prin_data, &status);
ERR_CHECK ("sec_rgy_pgo_get_by_name", status, error_status_ok, TRUE);

uuid_to_string (&prin_data.id, &string, &status);
ERR_CHECK ("uuid_to_string", status, uuid_s_ok, TRUE);
printf (" UUID: %s\n", (char *)string);
printf (" UNIX ID: %ld\n", (long)prin_data.unix_num);
printf (" Full name: '%s'\n", (char *)prin_data.fullname);

printf ("\nGetting group membership list...\n");

sec_rgy_cursor_reset(&cursor_g);
ERR_CHECK ("sec_rgy_cursor_reset (g)", status, error_status_ok, TRUE);
```

```

count = 0;
while (1) {

    sec_rgy_pgo_get_members(rgy_site, sec_rgy_domain_person, prin_name,
        &cursor_g, max_groups, group_list, &num_groups, &total_num_groups,
        &status);
    if (status == sec_rgy_no_more_entries) break;
    ERR_CHECK ("sec_rgy_pgo_get_members", status, error_status_ok, TRUE);

    if (count >= total_num_groups) {
printf ("bailing, hmm-ha\n");
        break;
    }
    if (!count) {
        printf ("%s is a member of %d groups:\n",
            (char *)input_name, (int)total_num_groups);
    }

    for (i = 0; i < num_groups; i++) {
        printf ("  Group %d: '%s'\n", ++count, group_list[i]);
    }
}

```

Retrieval of the user's existing password might be the most difficult aspect because it is stored in the DCE registry using a one-way encryption algorithm. This requires that accounts be created in the NFSv4 environment using new, possibly randomly generated passwords based on the site's existing security policy. Another option is to write a user-accessible utility that queries (without echoing it to the terminal) the user for their existing password, uses a DCE function, such as **dce_login -noexec** (for a script) or the **sec_()** family of functions (for a C program), to test the correctness of the password, and then uses the clear text version of the password to set the user's new Kerberos password. Users should be required to change this password upon initial login; most Kerberos V5 implementations set a password change flag on new user accounts by default to ensure that users comply with this requirement.

Writing a script involves the following steps:

1. Retrieve the user data from the results of a **sec_salvage_db** or **dcecp -c user show** command.
2. Parse the results for fields such as UID, group, or shell that are required in the NFSv4 environment. Note that group information is irrelevant in the context of a files-based (non-LDAP) Kerberos V5 subsystem, so the required input fields might vary by implementation.

Example 10-6 shows an example Perl script to retrieve account data from the **dcecp -c user show** command.

Example 10-6 Sample Perl script to retrieve account data

```
use Cwd; # module for finding the current working directory
use Getopt::Long qw(GetOptions);

my $user_principal = "";
my ($item, $uid, $group, $home, $shell) = "";

## main logic

GetOptions ( 'user=s' => \$user_principal);

if (open (USER_DATA, "dcecp -c user show $user_principal 2>&1 |") ) {
while (my $input_line = <USER_DATA> ) {
$input_line =~ s/^\{|\}$//g;
    ($item, $uid) = split " ", $input_line if ($input_line =~ "^uid");
    ($item, $group) = split " ", $input_line if ($input_line =~ "^group");
    ($item, $home) = split " ", $input_line if ($input_line =~ "^home");
    ($item, $shell) = split " ", $input_line if ($input_line =~ "^shell");
    }
}

print "User: $user_principal\n";
print "UID: $uid\n";
print "shell: $shell\n";
print "group: $group\n";

exit(0);
```

3. Using the data obtained in step 2, create accounts in the destination authentication system (for example, Kerberos V5). This step involves two possible scenarios:
 - If the system on which the script is running has been configured for concurrent access to both DCE and the destination authentication subsystem, the same script can simply generate appropriate commands (for example, **kadmin -q addprinc [...]**) and execute them.
 - Otherwise, the script can use the data to write a shell script containing the same set of **kadmin** commands. This file must then be copied to a system where the new authentication system is configured, after which it can be executed to create the new accounts.

Important: Execution of account-creation commands such as **kadmin** on the command line requires use of either the **-p <administrative password>** or **-C <keytab file>** parameters. The first requires that the user executing the script have knowledge of this password, while the second involves storing the key in a user-accessible file. Use both with extreme caution due to the security ramifications of exposing this password to non-administrative personnel.

Accounts can also be migrated to an Active Directory server, which also uses Kerberos V5 with LDAP. Example 10-7 provides a fragment from a script showing the basic steps required to create an account in Active Directory using Perl. This requires the Win32::OLE module, which can be obtained for free from repositories such as the Comprehensive Perl Archive Network (CPAN) at:

<http://www.cpan.org>

Example 10-7 Script fragment: Steps needed to create accounts in Active Directory

```
if (Win32::OLE->GetObject("LDAP://CN=$princ_name, CN=Users, $ldap") == "") {

    my $objParent = Win32::OLE->GetObject("LDAP://CN=Users, $ldap");
    my $objUser = $objParent->Create("User", "cn=$princ_name");
    $objUser->Put("sAMAccountName", "$princ_name");
    $objUser->SetInfo;
    ## only populate name information if we managed to parse
    ## the user's given and surname data earlier.
    if ( ($first_name) && ($last_name) ) {
        $objUser->Put("givenName", "$first_name");
        $objUser->Put("sn", "$last_name");
    }
    $objUser->SetPassword("$passwd");
    $objUser->SetInfo;
    $objUser->Put("userPrincipalName", "$princ_name@$ldap");
    $objUser->Put("userAccountControl", 512);
    $objUser->{AccountDisabled} = 0;
    $objUser->SetInfo;

} else { print "User $princ_name already exists...skipping\n"; }
```

Important: The use of this example requires administrative access to the Windows Active Directory database; earlier caveats regarding security apply in this case as well.

4. Optionally, the script can then disable the user's DCE account using a command such as **dcecp -c account modify <user_id> -acctvalid no**.

Again, this requires administrative or cell_admin-like access to the DCE registry, so its use might be limited by security concerns.

10.2.4 Authentication methods

Existing authentication practices should be used to provide a basis for the configuration of the new environment. For example, if DCE/DFS users commonly log in using traditional UNIX-style authentication methods, such as an entry in the /etc/passwd file, and only authenticate to DCE with **dce_login** when it becomes necessary to gain access to data located in DFS filesets, there might be no reason to enable integrated login in the NFSv4 environment. Conversely, sites that already make use of a single sign-on (SSO) methodology are unlikely to replace it with one requiring an initial UNIX login and a subsequent call to **kinit** or another post-login application to grant access to NFS-resident data requiring authentication. This is especially true if users require simultaneous authenticated access to data stored in multiple repositories (NFSv4 and Windows Server volumes, for example). Take into account the ease of use of an SSO solution, as well as a simpler management environment, when planning the migration.

The status of the current authentication method can be determined by examining, for example, the /etc/security/user file on an AIX 5L system for stanza entries such as:

```
SYSTEM = "DCE OR (DCE[FAILURE] AND AFS[SUCCESS])"
```

This entry denotes an initial attempt to validate a login using DCE; if failure occurs (due to an invalid password or the account not being present in the DCE cell, for example), the system will attempt a login through AFS. Failure of both methods results in a login failure. The same information can be found in the /etc/pam.conf file on a system using the Pluggable Authentication Model (PAM).

10.2.5 Additional considerations

As noted in Chapter 6, "Building an NFSv4 environment" on page 119, when eliminating existing services, you must take into consideration all resources or applications that rely on that architecture. For example, DFS is only one of potentially many DCE-based applications: Transaction processing monitors, printing subsystems, or location-brokering services might make use of the DCE RPC, security, or directory service. If such applications are in use, it might be necessary to retain the base DCE environment even after the DFS namespace has been fully decommissioned. This might require certain users to maintain accounts in both the DCE registry and Kerberos V5, for example. Concurrent operation of both services on the same machine requires special configuration

techniques, because the DCE security server also acts as a Kerberos V5 KDC and makes use of the standard port (88) when listening for requests.

If it becomes necessary to offer both the DCE security service and, for example, a Kerberos V5 client environment on one or more systems, the latter will need to be configured to operate using another port not currently allocated by other applications. The **krb5kdc** command is used to start a standard Kerberos V5 KDC, for example:

```
krb5kdc [ -d dbname ] [ -k keytype ] [ -M mkeyname ] [ -p portnum ] [ -m ] [ -r realm ] [ -4 v4mode ]
```

The **-p** parameter specifies which port will be used by the server. The **kdc** line in the **krb5.conf** file will also require modification; usually, it is displayed in the form **kdc = madrid.itsc.austin.ibm.com:88**, but the port number must match that used when starting the **krb5kdc** process. See the MIT Kerberos documentation or the *IBM Network Authentication Services Administrator's and User's Guide* (provided by the **krb5.doc.en_US** fileset) for additional information about the use of alternate ports when creating a new realm.

Using a DCE security server as a Kerberos V5 KDC

The DCE authentication service is based on an early release of Kerberos V5, and existing DCE security servers can act as master or slave KDCs for non-DCE users (for example, MIT Kerberos, IBM Network Authentication Services, or Active Directory). It is, therefore, possible to use an existing DCE environment to provide Kerberos services to clients and applications. However, the following caveats apply:

- ▶ It is not possible to create a standard slave KDC and migrate DCE security data to it. The DCE propagation model causes slave servers to request updates, pulling them from the master on a regular schedule. This differs from the “push” methodology used by Kerberos V5; in this case, the master KDC regularly copies the entire security database to its slave servers. The two are not compatible, and the slave KDC will never receive updates from the master DCE security server.
- ▶ A Kerberos V5 KDC master cannot be used to provide authentication services to DCE users. DCE accounts contain data such as group membership lists, home directories, and access limitations not found in a standard Kerberos V5 principal, so authentication attempts will fail.
- ▶ DCE creates the **krb5.conf** file in the **/etc** directory, while MIT Kerberos V5 and other implementations place it in **/etc/krb5**. DCE makes very little use of this file, but it might need to be edited in order to provide additional stanzas or other information required by non-DCE clients.

Figure 10-2 on page 237 shows an example DCE **krb5.conf** file.


```

[libdefaults]
    default_realm = dcerealm.itsc.austin.ibm.com
    default_keytab_name = /krb5/v5srvtab
    default_tkt_enctypes = des-cbc-crc
    tkt_lifetime = 4320
    default_tgs_enctypes = des-cbc-crc

[realms]
    dcerealm.itsc.austin.ibm.com = {
        kdc = server1.itsc.austin.ibm.com:88
        kdc = server2.itsc.austin.ibm.com:88
        kdc = server3.itsc.austin.ibm.com:88
    }

[domain_realm]
    test1.itsc.austin.ibm.com = dcerealm.itsc.austin.ibm.com

```

Figure 10-2 Example DCE krb5.conf file

10.3 ACL migration considerations

Special consideration must be given to the file system ACLs when planning a migration from DFS to NFSv4. DFS ACLs are POSIX based, while the NFSv4 protocol specification uses the Microsoft Windows ACL model. This means that a simple 1:1 translation from DFS to NFSv4 is not possible. This section presents a guide about how to migrate ACLs from DFS to NFSv4 while retaining as much functionality from the original DFS environment as possible. It might be valuable to review 7.5, “NFSv4 user authorization methods” on page 178 to ensure that the requirements of the NFSv4 ACL setup are understood before proceeding with this section. First, we review DFS ACLs and how they are evaluated. Then, we describe a mechanism for translating ACLs from DFS to NFSv4. Finally, we present examples showing how these translations can be automated using simple scripts.

Note: For more information about NFSv4 ACL support, see *AIX 5L Version 5.3 Security Guide*, SC23-4907, and *AIX 5L Differences Guide Version 5.3 Edition*, SG24-7463.

10.3.1 Understanding DFS ACL evaluations

DFS ACLs are evaluated in the following sequence:

1. user entries
2. group entries
3. other_obj
4. foreign_other
5. any_other

The ACL manager checks from the most specific ACLs to the least specific ACLs and stops at the first match that it finds for the requested permissions. So, if a user has both a user-level ACL entry and a group-level ACL entry, only the user-level entry will be evaluated. Figure 10-3 on page 239 shows the ACL checking process in flowchart format.

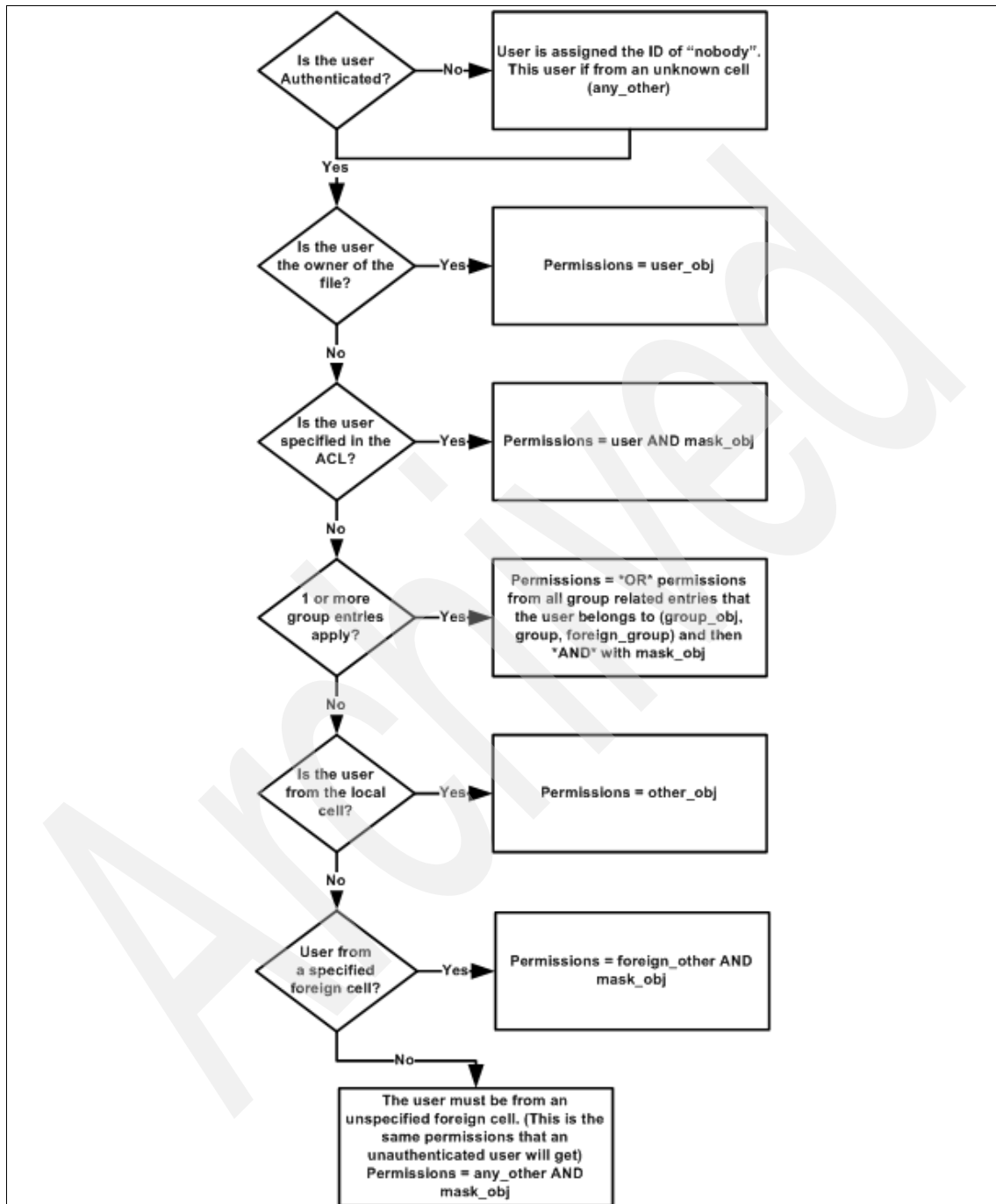


Figure 10-3 DFS ACL decision flow

10.3.2 DFS to NFSv4 ACL translation

One of the key differences between DFS and NFSv4 ACLs is the flow of ACL evaluation. DFS has a hierarchical method of evaluation for ACLs in which user_obj, then user+mask_obj, then group+mask_obj, then other_obj, and so on are evaluated to determine what permissions will be granted. This order of evaluation will be used regardless of the ACL order displayed in the ACL listing.

NFSv4 ACLs, however, depend on the order in which the ACLs appear in the listing to determine what ACL entry will be used to grant or deny requested permissions to the user. This key difference must be remembered when re-creating existing DFS ACL structures in NFSv4.

10.3.3 DFS and NFSv4 ACL comparisons

Table 10-1 lists ACL entries that can be used in NFSv4 to provide ACL permissions equivalent to those found in DFS.

Table 10-1 Equivalent ACL Entries in NFSv4s

Permission description	DFS ACL	NFSv4 ACL
Permission to read the data of the file	r	ra
Permission to list the contents of the directory	rx	rax
Permission to append to a file	w	wp
Permission to modify the file's data	rw	rwpa
Permission to add a new file to the directory	wix	wx
Permission to execute the file or traverse the directory	x	x
Permission to change the ACL of the file or directory	c	cC
Permission to delete files or subdirectories from within the directory	d	d

10.3.4 Example of DFS to NFSv4 ACL translation

We provide the following example of a DFS to NFSv4 ACL translation to illustrate a situation in which a user wants to preserve existing DFS home directory ACLs, allowing specific users and groups access to their files, after moving them to NFSv4.

Scenario: Home directory ACL migration

In this example, user “Joe” has migrated his DFS files to a new NFSv4 exported GPFS fileset. Joe had several ACLs in DFS that he wants to retain in his NFS environment. Example 10-8 shows original DFS home directory ACL structure.

Example 10-8 The original DFS home directory ACL structure for user Joe

```
# ls -ld joe
drwxrwxr-x  2 joe      sales          256 Aug 19 10:18 joe
#
# dcecp -c acl show joe
{mask_obj rwxcid}
{user_obj rwxcid}
{user alice r-x---}
{user keith rwx-id}
{group_obj rwxcid}
{group admins rwxcid}
{other_obj r-x---}
{any_other -----}
#
```

The following list provides an ACL-by-ACL breakdown of Example 10-8:

- {mask_obj rwxcid}

This will not have any impact on the NFSv4 translated ACLs because it has no restrictions and does not impact the DFS ACL list.
- {user_obj rwxcid}

Being the owner of this file, Joe wants to have full access to his NFSv4 ACL list. Remember that, by default, the owner of an NFSv4 file will always have permission to modify the ACLs of the file. The equivalent permissions in NFSv4 for full access for the owner will be:

s:(OWNER@): a rwpRWxDaAdCcs
- {user alice r-x---}

Joe wants Alice to retain her read access on the files in his directory. To grant this access, use the following ACE on Joe’s home dir:

u:alice: a rxa
- {user keith rwx-id}

Joe wants to retain Keith’s ability to modify/add/delete the files in Joe’s home directory. This can be accomplished with:

u:keith: a rwpRWxDaAds

This ACL allows all but control access on the directory. Note that the individual files and directories in Joe’s home directory can still maintain

their own ACL entries that still restrict Keith's access to them.

{group_obj rwxcid}

The group_obj ACL will apply to both the sales group that is displayed in the **ls -l** command and to any other groups that occur in the DFS ACL. In this case, admins. To give the sales group access to Joe's new home directory, we can change the group using **chgrp sales joe**.

Important: When changing the UNIX group of a file that has NFSv4 ACLs, you must be root *and* be a member of the group to which you are you are chgrp'ing. Therefore, in the previous example, you must be root *and* a member of sales to **chgrp** Joe's directory to sales. If you are not a member of sales, and you attempt to **chgrp sales joe**, you receive the message, **chgrp: joe: Operation not permitted**.

With the directory group ownership updated, the s:(GROUP@): ACL in the NFSv4 ACL list can be modified as follows:

```
s:(GROUP@):      a      rwpRWxDaAds
```

This ACL grants all but control access to the directory. Because there is an admin group that is also in the DFS ACL list, we add a second group entry to the ACLs:

```
g:admins:        a      rwpRWxDaAdCs
```

This gives full access on this directory to the admins group:

{other_obj r-x---}

The other_obj ACL applies to any authenticated DFS users that are not explicitly in any of the ACL entries in the list. In this case, read and pass-through should be granted to these users. There is no equivalent to this ACL in NFSv4 because AUTHENTICATED@ is not yet supported.

{any_other -----}

The any_other ACL applies to any users that do not fall into any users or groups that are in the ACL list. Because this is always processed last in DFS, the ACL should appear at the end of the ACL list to have the same affect in NFSv4:

```
s:(EVERYONE@):  d      rwpRWxDaAdCs
```

Example 10-9 on page 243 shows the resulting ACL list in NFSv4.

```
# aclget joe
*
* ACL_type   NFS4
*
*
* Owner: joe
* Group: sales
*
g:admins:      a      rwpRWxDaAdcCs
u:mary: a      rxa
u:keith:      a      rwpRWxDaAds
s:(OWNER@):   a      rwpRWxDaAdcCs
s:(OWNER@):   d      o
s:(GROUP@):   a      rwpRWxDaAds
s:(GROUP@):   d      wpWDACo
s:(EVERYONE@): d      rwpRWxDaAdcCso
#
```

Note: The order is important when creating these ACL entries. It is a good idea to keep admin groups that need extra or full access to the directories at the top of the list because they will then be evaluated first. In this case, the admins group is intentionally placed at the top of the list to ensure that all admins will have the appropriate access. Likewise, if the EVERYONE group in the previous ACL list appears before any other ACL entry, those trailing ACLs would never be processed under any circumstances. For more information about ACL processing see 7.5, “NFSv4 user authorization methods” on page 178.

Chapter 11, “Illustrated DFS migration” on page 247 presents additional information and hints about the methods by which existing DFS ACLs can be migrated to a new NFSv4 environment using Perl scripts. The appendixes also include the complete source code for these example scripts.

10.3.5 Data migration

Numerous options exist for accomplishing the actual migration of user and other data from DFS to the NFSv4 environment, but limitations might be encountered based on the type of migration and the hardware acquisition plans arrived at during the initial planning phase.

For example, if the site has opted for an all-at-once migration involving no new hardware, the only option for data migration involves creating offline backups on tape or other media, with subsequent restoration after the systems are reconfigured for an NFSv4 environment. If, however, the existing systems are

supplemented with additional hardware and storage space, user and other data can be copied across the network or even from disk to disk on an existing system.

Consider the following brief list of options:

- ▶ Tape backup, with restoration after the disks have been reformatted for JFS2 or GPFS and the NFSv4 environment has been established. Because this type of migration involves removing existing accounts, file ownership will be incorrect after the restoration has been accomplished unless UID and GID data is preserved across the migration. If accounts are created in the new environment using different UID/GID mapping, users might either be incapable of accessing their own files or might have inappropriate permissions on those owned by other users.

For example, user Bob's account was created under UID 95556 in the existing DCE security subsystem. His files are restored in the new environment, but this UID no longer maps to any account because UIDs were not preserved across the migration. An `ls -l` command on Bob's directory causes raw UID numbers to be displayed in the owner field rather than Bob's login name, and he is unable to access any of his data.

In another scenario, Bob's files are restored, but his UID of 95556 is assigned to user Jane instead. The result is that Jane effectively owns Bob's files; an `ls -l` command on Bob's directory shows Jane as the owner.

Obviously, the same rules apply regarding GID preservation.

- ▶ Networked (system-to-system) data migration using FTP or other tools. Again, file ownership at the destination must be closely monitored to ensure that it is correct for the new user account.

In both of these cases, problems can be avoided if new accounts are created using data extracted from the existing DCE security environment (for example, using the DCE LDAP migration tools or by extracting appropriate information from a `dcecp -c user show` command).

Alternately, if it is desirable to create a completely new set of accounts without preserving an existing UID/GID scheme, a clear mapping between old and new information will allow administrators to modify ownership of files after they have been copied to their destination directories. Using the previous example, the migration team simply creates a new account for Bob and uses the `chown` command to modify ownership after Bob's files have been moved to their new location. Automating this process using scripts can save a significant amount of time and effort.

- If some or all systems are configured to maintain both the existing DCE/DFS environment and NFSv4 mounts, you can use volume-to-volume local copying. File ownership is simpler to preserve in this case, as long as users can authenticate simultaneously in both environments.

This option also requires that administrators create the necessary file system skeleton, including the user's new home directory, before data migration can take place. This directory must have proper ownership and permissions so the user has the ability to copy data into it from the existing DFS directory tree.

A login session for a migrating user using the previous scenario involves the following steps:

- a. Log in to the new environment using the newly created account (for example, Kerberos, Network Authentication Services, or Active Directory).
- b. Use the **dce_login** command to gain credentials in the existing DCE cell.
- c. Use the **cp -R** command to recursively copy files from the DFS space to the newly created NFSv4 file system skeleton.
- d. Verify the proper ownership of files in the NFSv4 space.
- e. Optionally, use a script or the **aclput** command to alter any ACLs that need to be maintained. See the example ACL migration scripts provided in Appendix E, "Scripts and configuration files" on page 353 for tips about how such a utility can be implemented.

Illustrated DFS migration

This chapter presents an example of the steps required when migrating a DCE cell to a Kerberos V5/LDAP environment. We selected a user-by-user example for this scenario. The migration method presented here differs from that presented in the *IBM Distributed Computing Environment Version 3.2 for AIX and Solaris: DCE Security Registry and LDAP Integration Guide* and also provides additional details regarding data migration and ACL preservation. Sites are encouraged to use solutions appropriate for their particular circumstances.

We use the following migration sequence:

1. Configure the DCE cell. It is populated with 1500 users and 100 groups. Each user was added to 50 of the groups.
2. Configure the Kerberos V5 (KRB5)/LDAP environment.
3. A script is used to migrate DCE groups. User accounts are then migrated on a user-by-user basis using other scripts. GID and UID information is preserved.
4. A script is executed to capture existing ACLs on files in the DFS namespace.
5. User space directories and files are copied to the NFS namespace.
6. ACLs on these files are changed to NFSv4 style using the AIX 5L **aclconvert** command.
7. Last, ACLs are re-mapped to rough equivalents of their DFS values using the script and .acl files created in step 4.

We discuss the following topics in this chapter:

- ▶ Test environment
- ▶ Migrating the DCE cell to LDAP/KRB5
- ▶ Migrating user data

11.1 Test environment

The hardware and architecture used for these tests consists of a two-machine DCE 3.2 cell, with the name `dce_test.itsc.austin.ibm.com`. We use the following machines:

- ▶ `milan.itsc.austin.ibm.com` (DCE security and CDS master, DFS fileset server) running AIX 5L V5.2
- ▶ `angelina.itsc.austin.ibm.com` (DCE security replica), running AIX 5L 5.3 RML03

The target Kerberos V5/LDAP server is `pecos.itsc.austin.ibm.com` with AIX 5L V5.3 RML01. Its Kerberos realm name is `NFSV4REALM.IBM.COM`.

For the purposes of this scenario, a generic IBM Tivoli Directory Server environment was prepared. The `IBM.KRB.schema.ldif` file was loaded using the `ldapadd` utility so that integrated login can be enabled using the KRB5LDAP method.

We tested this configuration using the `mkuser` utility to ensure that it was possible to create users whose accounts properly resided in the LDAP repository.

11.2 Migrating the DCE cell to LDAP/KRB5

As discussed more fully in the *IBM Distributed Computing Environment Version 3.2 for AIX and Solaris: DCE Security Registry and LDAP Integration Guide*, the supported migration path uses a utility developed exclusively for IBM DCE V3.2. It permits an existing DCE security cell's data to be copied to an LDAP repository, after which a decision can be made to maintain a hybrid DCE/LDAP security and group environment or to decommission the existing DCE servers. This process must be conducted in a fixed order; otherwise, failures are likely to occur.

Presuming that the DCE cell has been fully upgraded to Version 3.2 at the most recent PTF level, perform the following steps:

1. Choose machines on which to run the LDAP master and slaves. Install and configure this product using its standard installation procedure. Note that LDAP slave servers must be configured on DCE security replicas that are to be migrated to LDAP.

Important: A migration server must be configured in order to accomplish the registry migrate procedure. This requires that both an existing security replica and an LDAP slave server be running concurrently on the same system.

2. In the case of IBM Tivoli Directory Server, use the **ldapadd** command to import the IBM.KRB.schema.ldif and IBM.DCE.schema.ldif files into LDAP. Other LDAP products might use different commands to achieve the same results.

Important: Although it might be possible to achieve the DCE to LDAP migration with IBM Network Authentication Services (NAS) configured into the LDAP environment, we do not recommend this due to possible conflicts with the NAS version of the IBM.KRB.schema.ldif file imported during configuration of NAS. As of this writing, the Kerberos schema files shipped with DCE and NAS are incompatible.

3. Run the **dcecp registry migrate** command on the security replica to create a migration slave.
4. For testing purposes, unconfigure the security replica in order to sever the migrated LDAP environment from the original DCE cell. Test the ability to log in as a Kerberos user using the former DCE data now stored in LDAP.

Migrating DCE groups and users

As noted earlier, the DCE **registry migrate** option was not used for this example scenario, because it was decided that a user-by-user migration was more appropriate for demonstration purposes. The built-in DCE option does not permit discrete users or groups to be migrated, so an alternative was developed using Perl scripts.

Tip: It is necessary to migrate groups before migrating user accounts if the **registry migrate** command is not used. AIX 5L **mkuser** commands fail if a non-existent group is specified. Therefore the following command will fail if, for example, the test group does not already exist in the KRB5LDAP registry:

```
mkuser -R KRB5LDAP pgrp=development groups=development,test myuser
```

Groups

The first script, `migrate_dce_groups_to_ldap.pl`, makes use of the **dcecp** commands **group catalog -simplename** and **group show -simplename**. The Perl source code for this script is provided in Appendix E, “Scripts and configuration files” on page 353, Example E-8 on page 376. The first **dcecp** command is used to generate a list of all groups in the DCE registry, and the second **dcecp** command to read the GID of each group. From this, it constructs a shell script called `dce_exported_groups.sh` containing **mkgroup** statements using the format shown in Example 11-1 on page 251.

Example 11-1 Format of mkgroup command output from migration script

```
#!/bin/sh

mkuser -R KRB5LDAP grpadm
mkgroup -R KRB5LDAP id=125 adms=grpadm Grp1
mkgroup -R KRB5LDAP id=126 adms=grpadm Grp2
mkgroup -R KRB5LDAP id=127 adms=grpadm Grp3
mkgroup -R KRB5LDAP id=128 adms=grpadm Grp4
mkgroup -R KRB5LDAP id=129 adms=grpadm Grp5
mkgroup -R KRB5LDAP id=130 adms=grpadm Grp6
mkgroup -R KRB5LDAP id=131 adms=grpadm Grp7
mkgroup -R KRB5LDAP id=132 adms=grpadm Grp8
mkgroup -R KRB5LDAP id=133 adms=grpadm Grp9
mkgroup -R KRB5LDAP id=134 adms=grpadm Grp10
```

Note that a user called grpadm is created and given administrative control over each group. This user ID is embedded in the script by the admin_user variable and can be changed before the script is executed if desired.

Tip: The root user cannot be given administrative control over groups listed in the KRB5LDAP registry, because root exists only in the files-based standard AIX 5L user registry.

As used, the DCE group migration Perl script also records built-in groups such as those shown in Example 11-2.

Example 11-2 Built-in DCE/AIX 5L groups captured by the migration script

```
mkgroup -R KRB5LDAP id=-2 adms=grpadm nogroup
mkgroup -R KRB5LDAP id=0 adms=grpadm system
mkgroup -R KRB5LDAP id=1 adms=grpadm daemon
mkgroup -R KRB5LDAP id=2 adms=grpadm uucp
mkgroup -R KRB5LDAP id=3 adms=grpadm bin
mkgroup -R KRB5LDAP id=4 adms=grpadm kmem
mkgroup -R KRB5LDAP id=6 adms=grpadm mail
mkgroup -R KRB5LDAP id=7 adms=grpadm tty
mkgroup -R KRB5LDAP id=12 adms=grpadm none
mkgroup -R KRB5LDAP id=18 adms=grpadm tcb
```

These groups should not be migrated because most will exist by default in the AIX 5L files-based registry (for example, the /etc/passwd file). Either they can be manually removed from the generated shell script prior to execution, or the migrate_dce_groups_to_ldap.pl script can be altered to bypass a specific list of such groups.

If certain groups already exist, the **mkgroup** command will report an error and continue. Example 11-3 shows these error messages.

Example 11-3 Possible error messages during group creation

```
3004-692 Error changing "id" to "7" : Account exists.
3004-692 Error changing "id" to "12" : Account exists.
3004-694 Error adding "acct-admin" : Name is too long.
3004-692 Error changing "id" to "200" : Account exists.
```

Additional development is required if an existing DCE site makes use of locally designated accounts with direct control over membership of discrete groups. In this case, output from the **dcecp ac1 show /.:/sec/group/group_name** command can be parsed in order to augment the previous commands with additional adms attributes. See Example 11-4.

Example 11-4 Example of an ACL on a group entry in DCE

```
# dcecp -c ac1 show /.:/sec/group/none
{unauthenticated r-t-----}
{user cell_admin rctDnfmM}
{user acct_mgmt rctDnfmM}
{group_obj r-t-----}
{group acct-admin rctDnfmM}
{other_obj r-t-----}
{any_other r-t-----}
#
```

Such additional command parsing is necessary if, for example, the site needs to give the acct-mgmt user an entry in the adms attribute. This step is left as an exercise for the reader.

Tip: Remember that, by default, AIX 5L user and group names cannot exceed eight characters in length.

User accounts

Next, a Perl script called `migrate_dce_users_to_ldap.pl` is executed to read user account data from the DCE registry (the source code is included in Appendix E, “Scripts and configuration files” on page 353, Example E-9 on page 378). Again, this generates a standard shell script called `dce_exported_users.sh`, containing appropriate **mkuser** commands for use on a system configured with the LDAP/Kerberos authentication subsystem. Example 11-5 on page 253 shows sample output; it was executed as:

```
perl migrate_dce_users_to_ldap.pl --prefix /nfs/dce
```


Example 11-5 Sample output from user migration script

```
#!/bin/sh
mkgroup -R KRB5LDAP migusr
mkuser -R KRB5LDAP uid=1138 pgrp=migusr groups=Grp20,Grp19,Grp18,Grp17,
Grp16,Grp15,Grp14,Grp13,Grp12,Grp11,Grp10,Grp9,Grp8,Grp7,Grp6,Grp5,Grp4,Grp3,Gr
p2,Grp1 home=/nfs/dce/home/home/testu30 testu30
mkuser -R KRB5LDAP uid=1137 geccos="Test User 29" pgrp=migusr groups=Grp20,
Grp19,Grp18,Grp17,Grp16,Grp15,Grp14,Grp13,Grp12,Grp11,Grp10,Grp9,Grp8,Grp7,Grp6
,Grp5,Grp4,Grp3,Grp2,Grp1 home=/nfs/dce/home/testu29 testu29
mkuser -R KRB5LDAP uid=1136 geccos="Test User 28" pgrp=migusr groups=Grp20,
Grp19,Grp18,Grp17,Grp16,Grp15,Grp14,Grp13,Grp12,Grp11,Grp10,Grp9,Grp8,Grp7,Grp6
,Grp5,Grp4,Grp3,Grp2,Grp1 home=/nfs/dce/home/testu28 testu28
mkuser -R KRB5LDAP uid=1135 geccos="Test User 27" pgrp=migusr groups=Grp20,
Grp19,Grp18,Grp17,Grp16,Grp15,Grp14,Grp13,Grp12,Grp11,Grp10,Grp9,Grp8,Grp7,Grp6
,Grp5,Grp4,Grp3,Grp2,Grp1 home=/nfs/dce/home/testu27 testu27
```

The script uses the first entry found in the DCE account's group list as the pgrp (primary group) attribute. However, if this value is "none," it is replaced with "migusr," which is created specifically to replace the DCE "none" group.

If users are to be migrated one at a time, the script can accept a DCE login ID as a parameter through the use of the **--user** flag. The latter option causes a single command, for example, **dcecp -c user show testu30**, to be executed and parsed. If the **--user** flag is not specified, the entire registry is read; in this case, the script uses the **dcecp -c account catalog -simplename** command to generate a list of user accounts, after which each returned account name is processed using the same **dcecp -c user show** command. The script also parses the home directory entry in the user's account, stripping off the DCE cell name and DFS /fs string if present.

Example 11-6 shows a complete example of a translation from a DCE account to a **mkuser** command. In this case, the Perl script is executed with the **--user** flag. The full command syntax is:

```
perl migrate_dce_user_to_ldap.pl --user testu30 --prefix /nfs/dce
```

Example 11-6 DCE registry entry for account testu30

```
# dcecp -c user show testu30
{fullname {Test User 30}}
{uid 1138}
{uuid 00000472-1671-21da-be00-000629b91ea4}
{alias no}
{quota unlimited}
{groups none Grp1 Grp2 Grp3 Grp4 Grp5 Grp6 Grp7 Grp8 Grp9 Grp10 Grp11 Grp12
Grp13 Grp14 Grp15 Grp16 Grp17 Grp18 Grp19 Grp20}
{acctvalid yes}
{client yes}
```

```

{created ../../dce_test.itsc.austin.ibm.com/cell_admin
2005-08-26-15:41:39.000-05:00I-----}
{description {}}
{dupkey no}
{expdate none}
{forwardabletkt yes}
{goodsince 2005-08-26-15:41:39.000-05:00I-----}
{group none}
{home ../../dce_test.itsc.austin.ibm.com/fs/home/testu30}
{lastchange ../../dce_test.itsc.austin.ibm.com/cell_admin
2005-08-30-09:58:48.000-05:00I-----}
{organization none}
{postdatedtkt no}
{proxiabletkt no}
{pwdvalid yes}
{renewabletkt yes}
{server yes}
{shell /bin/csh}
{stdtgtauth yes}
{usertouser no}
nopolicy
#

```

By default, the previous entry's home directory is translated to /home/testu30. However, because the **--prefix** flag is in use, the **mkuser** command produced by the Perl script uses the directory tree /nfs/dce/home/testu30. Additionally, the script detects the DCE group none and changes it to a new default (migusr) because the none group is unlikely to be necessary in the LDAP environment. The name of the default group can be altered by editing the script.

Example 11-7 shows the results obtained by running the script against the above account.

Example 11-7 Command produced during migration of user testu30

```

mkuser -R KRB5LDAP uid=1138 gecos="Test User 30" pgrp=migusr
groups=Grp20,Grp19,Grp18,Grp17,Grp16,Grp15,Grp14,Grp13,Grp12,Grp11,Grp10,Grp9,Grp8,Grp7,Grp6,Grp5,Grp4,Grp3,Grp2,Grp1 home=nfs/dce/home/testu30 testu30

```

The script can be expanded to capture additional data, such as the DCE account's expiration or creation time, if it is thought necessary to copy these fields intact to the LDAP database.

Review the resulting shell script for duplicate user IDs if the entire DCE registry is migrated (that is, the **--user** switch is not used), and, as with the group example shown earlier in the chapter, the presence of system-created or DCE-specific

accounts such as `cell_admin`, `uucp`, and `daemon`. Remove these prior to the execution of the shell script in the LDAP environment.

Example 11-8 shows example error messages that might occur during execution of the shell script in the KRB5LDAP realm.

Example 11-8 Errors encountered while creating users

```
3004-694 Error adding "testu28" : Account exists.
3004-694 Error adding "testuser25" : Name is too long.
3004-694 Error adding "testu23" : Account exists.
```

The **mkuser** command creates home directories only when the base directory path exists. In the previous example, the base path `/nfs/dce/home` must be present before executing the account-creation shell script; otherwise, errors will occur. Following execution, an `ls -l` of the newly created namespace shows the home directories and ownership information, as shown in Example 11-9.

Note: The account creation script, `dce_exported_users.sh`, must be executed under the root ID because **mkuser** is a privileged command. Otherwise, errors will be reported and the accounts will not be created.

Example 11-9 Directory listing following execution of the `dce_exported_users.sh` script

```
# ls -l /nfs/dce/home
total 0
drwxr-xr-x  2 testu1  migusr      256 Aug 30 11:21 testu1
drwxr-xr-x  2 testu10 migusr      256 Aug 30 11:21 testu10
drwxr-xr-x  2 testu12 migusr      256 Aug 30 11:21 testu12
drwxr-xr-x  2 testu13 migusr      256 Aug 30 11:21 testu13
drwxr-xr-x  2 testu14 migusr      256 Aug 30 11:21 testu14
drwxr-xr-x  2 testu15 migusr      256 Aug 30 11:21 testu15
[...]
drwxr-xr-x  2 testu15 migusr      256 Aug 30 11:21 testu30
#
```

You can test the new LDAP-based user account for `testu30` using the `lsuser -R KRB5LDAP testu30` command, as shown in Example 11-10.

Example 11-10 Fragment of user account listing for `testu30`

```
testu30 id=1138 pgrp=migusr groups=migusr,Grp1,Grp2,Grp3,Grp4,Grp5,Grp6,Grp7,
Grp8,Grp9,Grp10,Grp11,Grp12,Grp13,Grp14,Grp15,Grp16,Grp17,Grp18,Grp19,Grp20
home=/nfs/dce/home/testu30 shell=/usr/bin/ksh login=true su=true rlogin=true
telnet=true [...]
```

During testing, all elements present in the migration shell script were created properly. The only remaining issue is the password, which cannot be migrated through Perl scripts, and which must be changed in order to allow the user to log in to their migrated account. You can do this by using the script in Example E-2 on page 354, which uses the **kadmin** command to reset each password after new accounts have been created.

11.3 Migrating user data

In this section, we describe migrating user data.

11.3.1 Capturing existing ACLs in the DFS environment

Migrating ACLs from one environment to another is extremely difficult due to differences in mapping and handling between DFS and NFSv4. The DFS ACL set provides the “mask_obj” type that does not translate into a valid NFS-style ACL, so no attempt has been made to migrate this entry for purposes of this example. We used a Perl script called `migrate_dfs_acls_to_nfsv4.pl`; this script is in Example E-4 on page 361. It should by no means be considered a complete solution, but can be used as a basis for a more fully featured tool and customized for a location’s particular needs.

Moving ACLs involves the following steps:

1. A user (for example, `testu30`) whose files are to be copied logs in to the existing DFS environment using integrated login. The ACL migration script requires this, because it uses the `$HOME` environment variable to determine the base path with which it will start in the NFS environment. This requires the removal of the DFS `/.../cell_name/fs` prefix from subsequent **ac1put** commands.
2. The `migrate_dfs_acls_to_nfsv4.pl` script is run while the current working directory is the user’s `$HOME`:

```
perl migrate_dfs_acls_to_nfsv4.pl --prefix /nfs/dce
```

Example E-4 on page 361 provides the source code for this script. This script creates an `acl_data` subdirectory containing NFSv4-style `.acl` input files for each file and directory contained in the tree. It also creates a shell script, named `acl_testu30.sh` in the case of the `testu30` user, containing a series of AIX 5L **ac1put** commands referencing each file in the `acl_data` directory and its corresponding file. Example 11-11 on page 257 shows the example output.

Example 11-11 Commands in the `acl_testu30.sh` script

```
#!/bin/sh
aclput -i /nfs/dce/home/testu30/acl_data/.sh_history.acl
/nfs/dce/home/testu30/.sh_history
aclput -i /nfs/dce/home/testu30/acl_data/History.doc.acl
/nfs/dce/home/testu30/History.doc
aclput -i /nfs/dce/home/testu30/acl_data/afs_server.acl
/nfs/dce/home/testu30/afs_server
aclput -i /nfs/dce/home/testu30/acl_data/awk_example.acl
/nfs/dce/home/testu30/awk_example
```

An ACL input file constructed using the Perl script, as found in the `acl_data` directory, might look similar to the one shown in Example 11-12. This is a rough approximation of the ACL found on the matching file located in the DFS namespace. This shows the `.acl` input file generated by the Perl script for a file called `History.doc`, as well as the original DFS ACLs found in the DCE cell.

Example 11-12 Comparison of the ACL input file and original DCE/DFS ACL

```
$ cat acl_data/History.doc.acl
*
* ACL_type    NFS4
*
*
* Owner: testu30
* Group: Grp1
*
s:(OWNER@):   a      RAraWpXC
s:(GROUP@):   a      RArax
s:(EVERYONE@): a      RArax

# dcecp -c acl show ./History.doc
{user_obj rwx--}
{group_obj r-x---}
{other_obj r-x---}
#
```

11.3.2 Copying data from DFS to the NFS namespace

Next, the user's DFS-resident files, including the `acl_<username>.sh` file and the contents of the `acl_data` directory, are copied to NFS space using a standard AIX 5L `cp -R` command. This can also be accomplished using `tar` and FTP if desired. The result is that data formerly located in DFS under its `/home/<user_id>` directory tree is now in `/nfs/dce/home/<user_id>` and ready for ACL conversion.

The next step is to ensure that ACLs on the files in the `/nfs/dce` space are in NFS4 format, because they are likely to be in AIXC format by default. To check,

use the **aclget** command on a file in the user's new home directory, for example, **aclget .profile**. If it returns an AIXC-style ACL, simply use the **aclconvert** command recursively on the whole directory tree to convert all files to NFS4 ACLs. Example 11-13 shows an example.

Example 11-13 Using aclconvert to change user ACLs to NFS4 format

```
$ pwd
/nfs/dce/home/
$
$ aclconvert -R -t NFS4 testu30
$
```

After this step is successfully completed, the last step is to restore the ACLs saved in the `acl_data` directory.

11.3.3 Restoring ACLs on the copied data

The final step in ACL conversion involves the execution of the `acl_<username>.sh` script generated earlier in the process. For the example user `testu30`, the `./acl_testu30.sh` command is executed after the user has logged in to the LDAP/NFSv4 namespace. This traverses the entire directory and executes the **aclput** command on all files, using individual `acl_data/<filename>.acl` files as input.

Restriction: The supplied Perl scripts are not exhaustive in their conversion of existing DFS ACLs to NFS4 format. NFS4-style ACLs are very complex and behavior will be different from that found under DCE/DFS. The scripts are intended as a starting point for migration assistance only. Experimentation in a test setting will assist migrating sites in determining the appropriate ACL settings for their particular needs.

Example 11-14 shows an example of an ACL on a file in user `testu30`'s home directory before and after the ACL conversion.

Example 11-14 NFSv4 ACLs on file `awkt` before and after script execution

```
$ aclget awkt
*
* ACL_type   NFS4
*
*
* Owner: testu30
* Group: migusr
*
s:(OWNER@):      a      rwpRWxaAdcCs
```

```

s:(OWNER@):      d      o
s:(GROUP@):      a      rRxadcs
s:(GROUP@):      d      wpWACo
s:(EVERYONE@):   a      rRxadcs
s:(EVERYONE@):   d      wpWACo
$
$ ./acl_testu30.sh
$ aclget awk
*
* ACL_type   NFS4
*
*
* Owner: testu30
* Group: migusr
*
s:(OWNER@):      a      rwpRWxaAcC
s:(GROUP@):      a      rRxaA
s:(EVERYONE@):   a      rRxaA
$

```

Users can opt to delete the `acl_data` directory and the `acl_<username>.sh` file itself, because they will no longer be needed after the DFS to NFSv4 ACL conversion process has been completed.

Planning a migration from AFS

In this chapter, we discuss issues that must be considered when migrating from an AFS environment to an NFSv4 environment.

We discuss the following topics in this chapter:

- ▶ A broad overview of AFS
- ▶ Security differences between AFS and NFSv4
- ▶ Migrating AFS users to NFSv4
- ▶ Migrating AFS groups to NFSv4
- ▶ Comparing an AFS “cell” and an NFS “domain”
- ▶ File system semantics
- ▶ Building a namespace
- ▶ Migrating AFS data to NFSv4 servers
- ▶ Access control lists

12.1 A broad overview of AFS

A brief discussion of key Andrew File System (AFS) terms and concepts is useful before proceeding with details describing an AFS migration to NFSv4.

12.1.1 A distributed file system

AFS is a distributed file system that enables users to share and access files stored in a network of computers as easily as they access files stored on their local machines. The file system is called distributed for this reason; files can reside on many different machines, but are available to users on every machine.

12.1.2 Servers and clients

AFS stores files on a subset of the machines in a network, called *File Server* machines. File Server machines provide a file storage and delivery service, along with other specialized services, to another subset of machines in the network known as *client* machines. In a standard AFS configuration, clients provide computational power, access to files stored in AFS, and other general purpose tools to users seated at their consoles. There are generally many more client workstations than File Server machines. AFS File Server machines use a number of server processes, so called because each provides a distinct specialized service: one handles file requests, another tracks file location, a third manages security, and so on.

12.1.3 Cells

A cell is an administratively independent site running AFS. The cell's system administrators make many decisions about configuring and maintaining their cell in the way that best serves its users, without having to consult the administrators in other cells, for example, determining how many clients and servers to have, where to put files, and how to allocate client machines to users.

Figure 12-1 on page 263 shows an AFS cell.

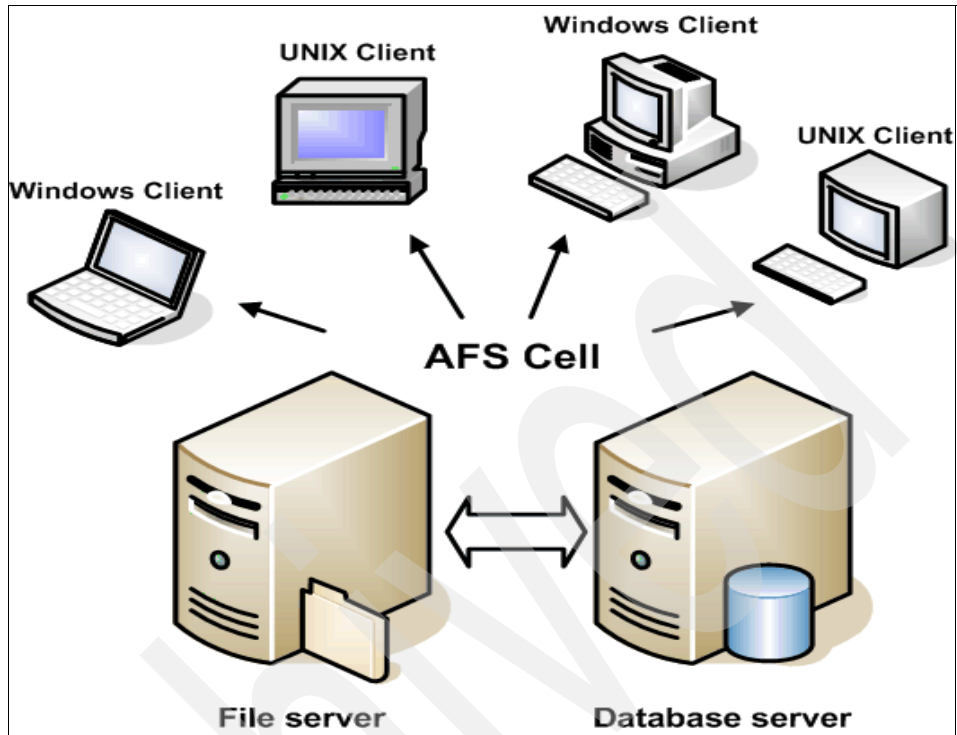


Figure 12-1 An AFS cell with servers and clients

Example 12-1 and Example 12-2 show the commands needed to determine the name of the cell to which a machine belongs and the database server names in each cell.

Example 12-1 Returns the name of the cell to which a machine belongs

```
# /usr/afs/bin/fs wscell
This workstation belongs to cell 'itsc.austin.ibm.com'
```

Example 12-2 Displays the database server machines in each cell

```
# /usr/afs/bin/fs listcell
Cell itsc.austin.ibm.com on hosts istanbul.itsc.austin.ibm.com
Cell test36.transarc.com on hosts cube.pittsburgh.ibm.com
afstest59.pittsburgh.ibm.com
```

12.1.4 Transparent access and the uniform namespace

Although an AFS cell is administratively independent, its local collection of files is organized in a manner that permits users from other cells to access its information. AFS enables cells to combine local file spaces into a *global* file space in a transparent manner. Users do not need to know anything about a file's location in order to access it. They must only know the path to the file, which looks the same in every cell. Therefore, every user at every machine sees the collection of files in the same way, meaning AFS provides a uniform namespace to its users.

Cross-cell sharing

Participating in the AFS global namespace makes a cell's local file tree visible to AFS users in foreign cells and makes other cells' file trees visible to local users. It makes file sharing across cells as easy as sharing within a cell. Making a file tree visible does not mean making it vulnerable. Participation in a global namespace is not mandatory. Some cells use AFS primarily to facilitate file sharing within the cell and are not interested in providing their users with access to foreign cells.

Tip: Assessing the impact of changes to existing cross-cell relationships is an essential component of the migration planning process.

12.1.5 Security: Mutual authentication and access control lists

Even in a cell where file sharing is especially frequent and widespread, it is not desirable that every user have equal access to every file. One way AFS provides adequate security is by requiring that servers and clients prove their identities to one another before exchanging information. This procedure, known as *mutual authentication*, requires that both server and client demonstrate knowledge of a shared secret (such as a password) known only to the two of them. Mutual authentication guarantees that servers provide information only to authorized clients and that clients receive information only from legitimate servers.

Users themselves control another aspect of AFS security. They determine who has access to the directories they own. For any directory a user owns, that user can build an access control list (ACL) that grants or denies access to the contents of the directory. An access control list pairs specific users with specific types of access privileges. There are seven separate permissions (*r1idwka*), and different IDs or groups can appear on an access control list.

Figure 12-2 on page 265 shows some AFS features.

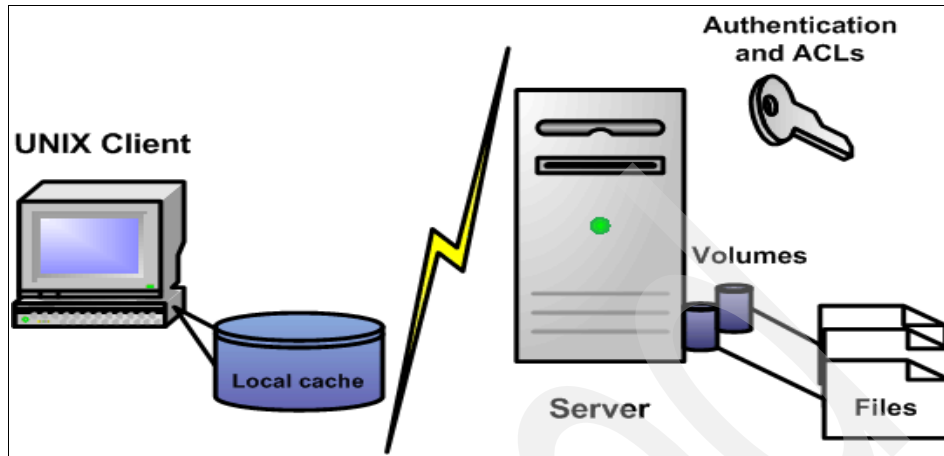


Figure 12-2 AFS features

12.1.6 Volumes

AFS groups files into *volumes*, making it possible to distribute files across many machines and yet maintain a uniform namespace. A volume is a unit of disk space that functions like a container for a set of related files, keeping them all together on one partition. Volumes can vary in size, but are smaller than a partition.

Volumes are important to system administrators and users for several reasons. Their small size makes them easy to move from one partition to another, or even between machines. The system administrator can maintain maximum efficiency by moving volumes to keep the load balanced evenly. In addition, volumes correspond to directories in the file space; most cells store the contents of each user home directory in a separate volume. Therefore, the contents of a directory move together when its containing volume moves, making it easy for AFS to keep track of a file's location at a given time. Volume moves are recorded automatically, so users do not have to keep track of file locations. Example 12-3 shows the output from the command to list volumes on an AFS server.

Example 12-3 List the volumes on an AFS server

```
# /usr/afs/bin/vos listvol istanbul.itsc.austin.ibm.com
Total number of volumes on server istanbul partition /vicepa: 28
eng                536870987 RW        420 K On-line
hr                  536870981 RW        420 K On-line
proj                536870969 RW         69 K On-line
projA               536870972 RW        426 K On-line
projB               536870975 RW        420 K On-line
root.afs            536870912 RW          5 K On-line
```

root.afs.readonly	536870913 R0	5 K On-line
root.cell	536870915 RW	14 K On-line
root.cell.readonly	536870916 R0	9 K On-line

12.1.7 Efficiency boosters: Replication and caching

AFS incorporates special features on server and client machines that help make it efficient and reliable. On server machines, AFS enables administrators to replicate commonly-used volumes, such as those containing binaries for popular programs.

Replication is a means of putting an identical, read-only copy (sometimes called a *clone*) of a volume on more than one File Server machine. The failure of one File Server machine housing the volume does not interrupt a user's work, because the volume's contents are still available from other machines. Replication also helps assure that one machine does not become overburdened with requests for files from a popular volume.

On client machines, AFS uses *caching* to improve efficiency. When a user on a client workstation requests a file, the *Cache Manager* on the client sends a request for the data to the *File Server* process running on the appropriate File Server machine. The user does not need to know which machine this is; the Cache Manager determines file locations automatically. The Cache Manager receives the file from the File Server process and adds it to the cache, an area of the client machine's local disk or memory dedicated to temporary file storage. Caching improves efficiency, because the client does not need to send a request across the network every time the user wants the same file. Network traffic is minimized, and subsequent access to the file is especially fast because the file is stored locally. AFS has a way of ensuring that the cached file stays up-to-date, a method known as *callback*.

12.2 Security differences between AFS and NFSv4

Both AFS and NFS incorporate several features to ensure that only authorized users gain access to data.

12.2.1 Security and authorization in AFS

AFS uses simple mutual authentication to verify user identities during the first part of the login procedure. In that case, the key is based on the user's password. Verification of the user's identity is accomplished when they provide their password during login. The Authentication Server grants the user a token as proof to AFS server processes that the user has authenticated.

All secure AFS transactions (except the first part of the login process) employ complex mutual authentication. When AFS client and server processes communicate, each requires the other to prove its identity during mutual authentication, which involves the exchange of encrypted information that only valid parties can decrypt and respond to. This helps create a secure environment in which to send cross-network messages.

In fulfilling these duties, the Authentication Server uses algorithms and other procedures based on Kerberos IV. The server also maintains the authentication database (kaserver.DB0), which records server encryption keys and an encrypted form of all user passwords. AFS sites can also use standard Kerberos authentication rather than the AFS Authentication Server.

AFS distinguishes between authentication and authorization checking. Authentication refers to the process of proving one's identity, while authorization checking involves the process of verifying that an authenticated identity is allowed to perform a certain action.

12.2.2 Security in NFSv4

NFSv4 uses the Sun remote procedure call (RPC) protocol to communicate over the network between the client and the server. The basic NFS security mechanisms are extended in NFSv4 through the mandated support of the *RPCSEC_GSS* RPC security flavor. *RPCSEC_GSS* is implemented at the RPC layer and is capable of supporting different security mechanisms. Examples include Kerberos V5 and public key-based mechanisms such as SPKM. NFSv4 requires that *RPCSEC_GSS* be provided as an available RPC security flavor. It mandates that the Kerberos V5, SPKM, and LIPKEY security mechanisms be supported for full protocol compliance. It still allows the support and use of other RPC security flavors such as *AUTH_SYS*. A key weakness of the *AUTH_SYS* flavor has always been the ability of a malicious hacker to forge and impersonate credentials quite easily.

Three levels of protection can be applied when using *RPCSEC_GSS* security with RPCs as they are transmitted over the network between server and client:

Authentication	Validates the identity of RPC sender.
Integrity	Validates that the contents of the RPC were not changed during transmission (includes authentication also).
Privacy	Prevents unauthorized viewing of data while it is in transit between client and server (includes authentication and integrity also).

Keep in mind that each increasing level of protection incurs a performance penalty. Each site must choose the minimum level that meets its data protection requirements.

12.2.3 Migration considerations

Note: The migration considerations mentioned in this book are for IBM AFS, but most of the discussion also holds true for OpenAFS.

Migrating existing authentication services from an AFS environment requires a great deal of planning, because the existing architecture might make use of more than one security data source. AFS cells might be using different authentication methods such as the AIX 5L integrated login or Solaris PAM to manage concurrent access to accounts located within the AFS kaserver, along with traditional flat files. If the AFS client machines use integrated login, users gain access to local file systems and AFS tokens in a single step. If an AFS-modified login utility is not in use, a standard UNIX password must be placed in the local password file of every client machine the user will use. The user logs in to the local file system only, and then must issue the `klog` or `pagsh` commands to authenticate with AFS. It is simplest if the passwords in the local password file and the Authentication Database are the same, but this is not required.

Note: Check the default stanza in the `/etc/security/user` file. If the `SYSTEM` line is similar to the following line, this means that AFS integrated login is enabled:

```
SYSTEM = "AFS OR AFS[NOTFOUND]AND compat"
```

Migrating to an NFSv4 environment might require that AFS and other accounts be loaded into an MIT Kerberos or IBM Network Authentication Services database in order to make use of, for example, the enhanced services and improved security of NFSv4. We provide an illustration of the various steps involved to set up and configure such an environment in 6.4, “IBM Tivoli Directory Server V5.2” on page 122.

Some AFS sites have already migrated to Kerberos V5 and no longer use the AFS kaserver. For them, it will be easier to migrate to NFS.

12.3 Migrating AFS users to NFSv4

Migration requires the creation of users and groups within an NFS environment that match those already present in the AFS cell. The list of AFS users can be obtained in many ways:

- The standard password files (`/etc/passwd` or equivalent) provide the list of users, names, login shell, and home directories. While adding AFS accounts, most AFS sites create an entry for the new user in the common shared password file. This file is then distributed to each machine that the user can log in to. This is a general practice for cells with large number of users because this centralized repository is easy to maintain. Example 12-4 shows a sample `/etc/passwd` file.

Example 12-4 Sample `/etc/passwd` file

```
root:!:0:0:::/usr/bin/ksh
daemon:!:1:1::/etc:
bin:!:2:2::/bin:
sys:!:3:3::/usr/sys:
adm:!:4:4::/var/adm:
uucp:!:5:5::/usr/lib/uucp:
guest:!:100:100::/home/guest:
user1:!:214:1:User One:/afs/itsc.austin.ibm.com/home/user1:/usr/bin/ksh
user2:!:215:1:User Two:/afs/itsc.austin.ibm.com/home/user2:/usr/bin/ksh
user3:!:216:1:User Three:/afs/itsc.austin.ibm.com/home/user3:/usr/bin/ksh
user4:!:217:1:User Four:/afs/itsc.austin.ibm.com/home/user4:/usr/bin/ksh
user5:!:218:1:User Five:/afs/itsc.austin.ibm.com/home/user5:/usr/bin/ksh
```

- You can also obtain a list of users using the **kas** command, as shown in Example 12-5.

*Example 12-5 Obtaining the list of users using the **kas** command*

```
# /usr/afs/bin/kas list admin
afs
admin
user1
user2
user3
user4
user5
```

- You can obtain a list of all accounts along with the matching user ID by using the **pts listentries -users** command, as shown in Example 12-6 on page 270.

Example 12-6 Obtaining the list of users using the pts command

```
# /usr/afs/bin/pts listentries -users
```

Name	ID	Owner	Creator
anonymous	32766	-204	-204
admin	1	-204	32766
user2	215	-204	1
user3	216	-204	1
user4	217	-204	1
user5	218	-204	1

After obtaining the list of users, the next step is to create the same users as principals in the Kerberos V5 subsystem. We illustrate the steps to do this in 13.5, “Migrating users to Kerberos and LDAP” on page 290.

The AFS Authentication Database stores user passwords converted with an encryption key, so retrieving the user’s existing password can be difficult. This requires that accounts be created in the NFSv4 environment using new, possibly randomly generated passwords based on the site’s existing security policy. Users should be required to change this password upon initial login; most Kerberos V5 implementations set a password change flag on new user accounts by default to ensure that users comply with this requirement. Therefore, at the time of first login, the users can reset their passwords back to the old AFS passwords, if they desired.

12.4 Migrating AFS groups to NFSv4

You can obtain the list of groups and their members using the **pts** command suite.

Example 12-7 shows how a list of groups can be obtained using the **pts** command.

Example 12-7 Obtaining the list of groups using the pts command

```
# /usr/afs/bin/pts listentries -groups
```

Name	ID	Owner	Creator
system:administrators	-204	-204	-204
system:anyuser	-101	-204	-204
system:authuser	-102	-204	-204
staff	-206	1	1
contract	-207	1	1
user11:proja	-208	224	1
friends	-209	229	1

As the previous example shows, a group can be a system group (system:anyuser), a regular group (user11:proja), or a prefix-less group (friends)

You can obtain a list of members for each of these groups, as shown in Example 12-8.

Example 12-8 Obtaining the list of members using the pts command

```
# pts membership friends
Members of friends (id: -209) are:
    user2
    user17
    user20
    user21
```

To display the groups a user or group owns, use the command shown in Example 12-9.

Example 12-9 Retrieving the owners for each group

```
# /usr/afs/bin/pts listowned user11
Groups owned by user11 (id: 224) are:
    user11:proja

# /usr/afs/bin/pts listowned user16
Groups owned by user16 (id: 229) are:
    friends
```

These groups must be created within the IBM Tivoli Directory Server LDAP back end. In Chapter 13, “Illustrated AFS migration” on page 285 (see Example 13-13 on page 295), we provide an example of this using the **mkgroup** command.

Other utilities in AFS, such as **prdb_check**, can provide more detailed information. This utility emits extended information about the AFS users and groups, as shown in Example 12-10.

Example 12-10 Sample output from the prdb_check utility

```
# /usr/afsws/etc/prdb_check -entries /usr/afs/db/prdb.DB0
Entry at 66752: flags 0x80, id 1i, next 0.
c:08/10 15:03:28 a:08/10 15:03:49 r:time-not-set n:time-not-set
ids -204
hash (id 0 name 0). Owner -204i, creator 32766i
quota groups 20, foreign users 20. Mem: 1, inst: 0
Owned chain 78464, next owned 0, inst ptrs(0 0 0).
Name is 'admin'
.
.
```

```
Entry at 71936: flags 0xc0, id 215i, next 0.  
c:08/16 11:05:15 a:08/17 11:29:14 r:time-not-set n:time-not-set  
ids -206 -209  
hash (id 0 name 0). Owner -204i, creator 1i  
quota groups 20, foreign users 20. Mem: 2, inst: 0  
Owned chain 0, next owned 0, inst ptrs(0 0 0).  
Name is 'user2'
```

Important: Some existing AFS cells have already migrated to Kerberos V5 and will not need to perform the steps described in this section.

12.5 Comparing an AFS “cell” and an NFS “domain”

An AFS *cell* is used as an administrative domain for configuring servers and clients. NFSv4 uses a similar concept known as a *domain*. For migration purposes, the AFS cell can effectively be transformed into an NFS domain. All clients and servers in that domain can then be configured.

We provide the steps to configure an NFS domain in 13.3, “Setting the NFS domain to the AFS cell name” on page 288.

12.6 File system semantics

In this section, we discuss file system semantics.

12.6.1 AFS implements save on close

When an application issues the UNIX *close* system call on a file, the Cache Manager performs a synchronous write of the data to the File Server that maintains the central copy of the file. It does not return control to the application until the File Server has acknowledged receipt of the data. When an application issues the UNIX *write* system call, the Cache Manager writes modifications to the local AFS client cache only. If the local machine crashes or an application program exits without issuing a close, it is possible that the modifications will not be recorded in the central copy of the file maintained by the File Server.

The implication is that if an application’s Save option invokes the write system call rather than close or fsync, the changes are not necessarily stored permanently on the File Server machine. Most application programs issue a save during close operations, as well as when they finish handling a file and on application exit.

12.6.2 Difference between AFS and NFS

An AFS file system provides an open-close (session) semantics and NFS follows the UNIX single-site (read/write) semantics. The UNIX semantics imply that in a centralized UNIX system, if a process modifies a file, other processes see the new data on the next read system call. Another way of saying this is that single-site semantics guarantee cache consistency at the read and write system call granularity, and session semantics do so at open and close system call level. This difference is important for any applications using AFS to store their data.

12.7 Building a namespace

NFSv4 offers a large variety of features when building the namespace. These features are as follows:

- ▶ pseudo root
- ▶ exname
- ▶ referrals
- ▶ replication

12.7.1 Pseudo file system

We discuss this feature in detail in 5.3, “Pseudo file system” on page 87.

Use of this NFSv4 feature allows the creation of an exported namespace similar to that provided by AFS. All AFS clients perform a single mount of the shared /afs directory tree. This provides the same namespace on all AFS clients. Similarly, all NFS clients perform a single mount of the NFS server's root export using the same directory (for example, /nfs) in order to access the same hierarchical namespace. This provides a simple, one-to-one mapping easily understandable to existing AFS users.

12.7.2 External namespace (exname)

This feature is not part of the NFSv4 RFC, but rather is an AIX 5L implementation-specific option. The **exname** option extends the pseudo file system concept. The external name in the /etc/exports file must begin with the nfs root name. But, an **exname** export does not need to correspond to the server's root.

The **exname** option is useful when we do not want to expose a server's file system tree to the client.

For more details about this option, see the IBM Redbook *Securing NFS in AIX: An Introduction to NFS V4 in AIX 5L Version 5.3*, SG24-7204. You can view or download this book from the following location:

<http://www.redbooks.ibm.com/abstracts/sg247204.html>

12.7.3 Referrals and replication

Both AFS and the NFSv4 protocol permit the distribution of data across multiple servers in a manner that makes its actual location transparent to users of that data. Several features assist with this process. The first is a global namespace feature called a *referral*. The second is a means of specifying locations where copies of data can be found, which is called a *replica*.

Figure 12-3 shows that the proj directory actually is located on another server. A referral is created in the namespace of a server and location information is attached to it. The primary server redirects operations to the server specified in the location information. Although this sounds simple, it provides very powerful capabilities for the location and administration of data. There is no restriction on the number of referrals that can be created within either the server's NFSv4 pseudo space or within an exported file system.

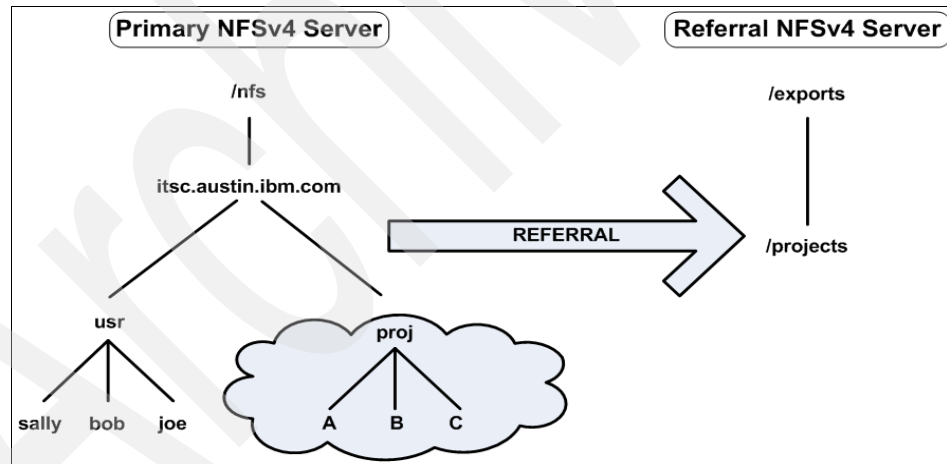


Figure 12-3 Example of a namespace using referral

The concept of referral helps provide location transparency, which is also a key feature of AFS. One of the features that makes AFS easy to use is that it provides transparent access to files in a cell's file space. Users do not need to know which File Server machine stores a file in order to access it; they simply provide the file's path, which AFS invisibly translates into a machine location. In addition to transparent access, AFS also creates a uniform namespace—a file's path is

identical regardless of which client machine on which the user is working. The cell's file tree looks the same when viewed from any client, because the File Server machines store all files centrally and present them in an identical manner to all clients.

Similarly, NFSv4 users need do not need to know where the data actually resides. A single mount of the NFS primary sever provides access to the entire namespace.

Figure 12-4 shows that the users directory is replicated. If the primary server fails or becomes inaccessible, clients will still be able to access directories by failing over to the replicated server.

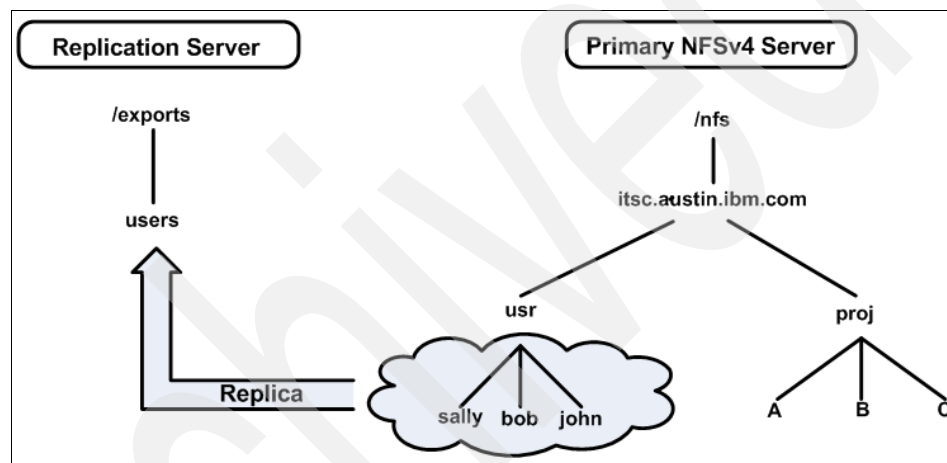


Figure 12-4 Example of a namespace with replication

This is similar to AFS replication. In AFS, a read-only volume is a copy of the read/write source volume and can exist at multiple sites (a site is a particular partition on a particular File Server machine). As the name suggests, a read-only volume's contents do not change automatically as the read/write source changes, but only when an administrator issues the **vos release** command. For users to have a consistent view of the AFS file space, all copies of the read-only volume must match one another as well as their read/write source.

We provide a detailed discussion about replication in NFSv4 in 3.3.3, "Replication" on page 36. We address examples of both the read-only (ro) and read/write (rw) replication, alongside the challenges of keeping the data synchronized on multiple sites. The administrator needs to decide which replication type is appropriate for their site. For example, replicating database files located within the NFS namespace might be advisable, but maintaining consistency across multiple copies might be challenging.

12.8 Migrating AFS data to NFSv4 servers

Each site must decide on an appropriate strategy for migration and ensure hardware availability before commencing data migration. The data in an AFS cell can be broadly classified into two types:

- ▶ User data

This consists of home directories, owned by individual users, which are used for holding individual personal data. Typically, the owner has full control over their own directory tree. When required, access to this data can be given to others. The concept of groups and ACLs helps effectively share this data. One method for migrating this data involves the use of commands such as **tar** and **cpio**. Depending on the type of migration selected, administrators might move this data or allow each user to be responsible for migrating their own files.

- ▶ Common/shared data

As the name suggests, this material is often accessed by multiple users or groups. Group owners are responsible for setting access controls on this data. Often, it is logically divided into directories based on project names, departments, or useful tools, binaries, or source code. Many sites run complex applications for creating and managing such data over AFS.

12.8.1 Migration options

Numerous options exist for accomplishing the migration of user and other data from AFS to the NFSv4 environment, but limitations might be encountered based on the type of migration and hardware acquisition plans arrived at during the initial planning phase.

For example, if the site has opted for an all-at-once migration involving no new hardware, the only option for data migration involves creating offline backups on tape or other media, with subsequent restoration after the systems are reconfigured for an NFSv4 environment. If, however, the existing systems are supplemented with additional hardware and storage space, user and other data can be copied across the network or even from disk to disk on an existing system.

Consider the following brief list of options:

- ▶ Tape backup, with restoration after disks have been reformatted for JFS2 or GPFS and the NFSv4 environment has been established. Because this type of migration involves removing existing accounts, file ownership will be incorrect after restoration has been accomplished unless UID and GID data is preserved across the migration. If accounts are created in the new environment using different UID/GID mapping, users might either be incapable of accessing their own files or might have inappropriate permissions on those owned by other users.

For example, user John's account was created under UID 1001 in the existing AFS security subsystem. His files are restored in the new environment, but this UID no longer maps to any account because UIDs were not preserved across the migration. An `ls -l` on John's directory causes raw UID numbers to be displayed in the owner field rather than John's login name, and he is unable to access any of his data.

In another scenario, John's files are restored, but his UID of 1001 is assigned to user Jane instead. The result is that Jane effectively owns John's files; an `ls -l` of John's directory shows Jane as the owner.

The same rules apply regarding GID preservation.

- ▶ Network (system-to-system) data migration using FTP or other tools. Again, file ownership at the destination must be closely monitored to ensure that it is correct for the new user account.

In both the previous cases, problems can be avoided if new accounts are created using data extracted from the existing AFS environment. Alternately, if it is desirable to create a completely new set of accounts without preserving an existing UID/GID scheme, a clear mapping between old and new information will allow administrators to modify ownership of files after they have been copied to their destination directories. Using the previous example, the migration team simply creates a new account for John and uses the `chown` command to modify ownership after John's files have been moved to their new location. Automating this process using scripts can save a significant amount of time and effort.

- ▶ The AFS file tree can be checked to see how the various volumes are organized. That way, we get a better idea about how we want to structure this data into NFS. For example, we might have to move data in certain AFS volumes/directories to multiple NFS servers due to disk space limitations. Better load balancing will be done if we use multiple NFS servers, and then create different attach points to get a combined namespace.

12.8.2 NFS/AFS Translator

Because the complete migration process might require significant time, data might reside partially in AFS and partially in NFS. You can use the NFS/AFS Translator to provide access to AFS-resident data by NFS-only clients.

The NFS/AFS Translator enables users on NFS client machines to access the AFS file space as though they are working on an AFS client machine. An NFS/AFS translator machine (or simply translator machine) is a machine configured as both an AFS client and an NFS server.

The AFS client functionality enables such a system to access AFS file space. The Cache Manager requests and caches files from AFS File Server machines, and can even maintain tokens for NFS users if configuration changes have been made to enable NFS users to authenticate to AFS.

Its NFS server functionality makes it possible for the translator machine to export the AFS file space to NFS client machines. When a user on an NFS client machine mounts the translator machine's /afs directory (or one of its subdirectories if that feature is enabled), access to AFS is immediate and transparent. The NFS client machine does not require AFS software.

The NFS/AFS Translator is available on a limited number of platforms and supports only NFSv3. But because NFSv3 and NFSv4 can coexist, this can be a useful tool while migrating data.

12.9 Access control lists

Special consideration must be given to file system ACLs when planning a migration from AFS to NFSv4. The NFSv4 ACLs are similar to Windows NTFS ACLs, but they are not identical. The developers of the NFSv4 standard chose the Windows ACL model over POSIX ACLs because the Windows ACL model is richer and more widely deployed.

This section discusses AFS ACLs and attempts to compare them with NFS ACLs. A simple 1:1 translation from AFS to NFSv4 ACLs is not possible. In addition, we provide an example showing how an AFS ACL can translate to an NFS ACL. These translations can be automated using simple scripts, an example of which is illustrated in 13.8, "Migrating ACLs" on page 300.

12.9.1 AFS ACL permissions

Functionally, the seven standard ACL permissions fall into two groups: one that applies to the directory itself and one that applies to the files it contains:

► The four directory permissions

The permissions in this group are meaningful with respect to the directory itself:

- The **l** (lookup) permission: This permission functions as something of a gate keeper for access to the directory and its files, because a user must possess it in order to exercise any other permissions. In particular, a user must have this permission to access anything in the directory's subdirectories, even if the ACL on a subdirectory grants extensive permissions. This permission enables a user to issue the following commands:
 - The **ls** command to list the names of the files and subdirectories in the directory
 - The **ls -ld** command to obtain complete status information for the directory element itself
 - The **fs listacl** command to examine the directory's ACL
- The **i** (insert) permission: This enables a user to add new files to the directory, either by creating or copying, and to create new subdirectories. It does not extend into any subdirectories, which are protected by their own ACLs. Because these are directory-level permissions, the **i** (insert) permission does not control adding data to a file, but rather creating a new file or subdirectory.
- The **d** (delete) permission: This enables a user to remove files and subdirectories from the directory or move them into other directories (assuming that the user has the **i** permission on the ACL of the destination directory).
- The **a** (administer) permission: This enables a user to change the directory's ACL. Members of the `system:administrators` group implicitly have this permission on every directory (that is, even if that group does not appear on the ACL). Similarly, the owner of a directory implicitly has this permission on its ACL and those of all directories below it that he or she owns.

► The three file permissions

The three permissions in this group are meaningful with respect to files in a directory, rather than the directory itself or its subdirectories:

- The **r** (read) permission: This enables a user to read the contents of files in the directory and to issue the **ls -l** command to start the file elements.

- The w (write) permission: This enables a user to modify the contents of files in the directory and to issue the **chmod** command to change their UNIX mode bits.
- The k (lock) permission: This enables the user to run programs that issue system calls to lock files in the directory.

AFS also uses the UNIX mode bits as follows:

- ▶ The initial bit is used to determine the element's type, for example, a - (dash) for a file or the letter d for a directory.
- ▶ It does not use any of the mode bits on a directory. For a file, only the first (owner) set of bits interacts with the ACL entries.
- ▶ If the first r mode bit is not set, no one (including the owner) can read the file, no matter what permissions they have on the ACL. If the bit is set, users also need the r (read) and l permissions on the ACL of the file's directory to read the file.
- ▶ If the first w mode bit is not set, no one (including the owner) can modify the file. If the w bit is set, users also need the w and l permissions on the ACL of the file's directory to modify the file.
- ▶ There is no ACL permission directly corresponding to the x mode bit, but to execute a file stored in AFS, the user must also have the r and l permissions on the ACL of the file's directory.

12.9.2 NFS ACL permissions

For a detailed listing of the NFSv4 ACLs, see 5.4, “NFSv4 ACLs” on page 90.

12.9.3 Detailed comparison of AFS and NFS ACLs

The NFS ACLs are set at the file level. Therefore, access to each file in a directory can be controlled with a different ACL. An AFS ACL instead protects all files in a directory in the same way. If a certain file is more sensitive than others, we need to store it in a directory with a more restrictive ACL.

There is no exact match between the NFS and AFS ACLs. You can use the information in Table 12-1 on page 281 as a reference for replacing existing ACLs.

Table 12-1 Comparing AFS and NFSv4 ACLs

AFS	NFSv4
r	r
l	racx
i	wp
d	dD
w	w
k	w
a	C

No direct correlation exists between NFSv4 ACL bits and the AFS ACL k. The NFS ACL w is a reasonably close approximation.

Table 12-2 lists some typical operations for which specific permissions are required for files and directories. The table provides entries that can be used in NFSv4 to provide ACL permissions equivalent to those found in AFS.

Table 12-2 Equivalent ACL entries in NFSv4s

Permission description	AFS ACL	NFSv4 ACL
Permission to read the data of the file	rl and the UNIX mode r bit set for the owner	ra
Permission to list the contents of the directory	l	rax
Permission to append to a file	wl and the UNIX mode w bit set for the owner	wp
Permission to modify the file's data	wl and the UNIX mode w bit set for the owner	rwpa
Permission to add a new file to the directory	i	wx
Permission to execute the file or traverse the directory	rl	x
Permission to change the ACL of the directory	a	cC
Permission to delete files or subdirectories from within the directory	d	dD

12.9.4 Example of an AFS to NFS ACL conversion

Example 12-11 shows what a sample AFS ACL typically looks like.

Example 12-11 Sample AFS ACL

```
# /usr/afs/bin/fs la .
Access list for . is
Normal rights:
  friends rlidw
  staff rlidwka
  system:administrators rlidwka
  system:authuser rl
  user1 rlidwka
```

When converted to an NFS ACL, it looks similar to the sample shown in Example 12-12.

Example 12-12 Sample AFS ACL converted to an NFS ACL

```
# aclget filea
*
* ACL_type   NFS4
*
*
* Owner: root
* Group: system
*
u:user1:      a      rwpxDadcC
g:staff:      a      rwpxDadcC
g:friends:    a      rwpxDadc
s:(OWNER@):   a      rwpxDadcC
```

Some observations about this conversion:

- ▶ As the example shows, the AFS groups friends and staff are now being referenced with an NFS ACE IDENTITY_type value of g (g:staff and g:friends). Usually, AFS ACLs have multiple group entries on each ACL. These must be translated into multiple g group entries. The original AFS rlidwka permissions were converted using the Table 12-1 on page 281.
- ▶ The user who owns this file is user1. The NFS ACE IDENTITY_type value of u (user) is used to represent this.

- ▶ Note that no attempt was made to convert the `system:administrators` and `system:authuser` directly to NFS. These AFS system groups have special meanings that do not translate directly to NFSv4 ACLs; migrating these ACLs to NFS requires extra care and planning. One possibility is creating a special NFS group to receive the AFS “system” groups. For example, create a group called `admgrp` and transfer all AFS `system:administrators` group ACLs to it.
- ▶ Setting negative permissions in AFS ACLs is generally unnecessary and not recommended. But if such entries exist, the NFS `ACE_TYPE` value of `d` (deny access) can be used.
- ▶ The order is important when creating these ACL entries. It is a good idea to keep admin groups that need extra or full access to the directories at the top of the list because they will then be evaluated first.

For more information about NFS ACL processing, see 5.4.1, “NFSv4 ACLs: ACL evaluation” on page 95. Chapter 13, “Illustrated AFS migration” on page 285 presents additional information and hints about methods by which existing AFS ACLs can be migrated to a new NFSv4 environment using Perl scripts. Appendix E, “Scripts and configuration files” on page 353 provides the source code for these example scripts.

Illustrated AFS migration

This chapter describes the migration of a sample AFS cell to an NFS environment.

We discuss the following topics in this chapter:

- ▶ Introduction
- ▶ Existing AFS cell setup
- ▶ Setting the NFS domain to the AFS cell name
- ▶ Setting up the KRB5/LDAP environment
- ▶ Migrating users to Kerberos and LDAP
- ▶ Migrating group information
- ▶ Migrating data
- ▶ Migrating ACLs
- ▶ Accessing the migrated data from NFSv4 clients

13.1 Introduction

This chapter presents the sequence of events used while migrating a test AFS cell to an NFS environment. The test environment consists of an AFS cell with the cell name `itsc.austin.ibm.com`. It is composed of a single AIX 5L V5.2 machine, `istanbul.itsc.austin.ibm.com`, running AFS V3.6. The target KRB5/LDAP server is `pecos.itsc.austin.ibm.com` with AIX 5L V5.3. Its Kerberos realm name is `NFSV4REALM.IBM.COM`. The same machine was also used as an NFS server, and the data from AFS was moved onto it. The machine `sabine.itsc.austin.ibm.com` was used as an NFS client to access the migrated data.

We use the following migration sequence:

1. Configure the AFS cell. It is populated with multiple users and groups. Users were added to different regular and prefix-less groups.
2. Configure the KRB5/LDAP environment so that NFSv4 can use `RPCSEC_GSS` for authentication.
3. Various scripts and commands are used to migrate users and groups from AFS to the KRB5/LDAP server.
4. A script is executed to capture existing ACLs on files in the AFS namespace. This generates `.acl` files containing the ACLs for each file.
5. User space directories and files are copied to the NFS namespace.
6. ACLs on these files are changed to NFSv4 style, using the AIX 5L `aclconvert` and `aclput` commands.
7. Lastly, the AFS ACLs are mapped to rough equivalents of their NFS values using the script and `.acl` files created in step 4.

13.2 Existing AFS cell setup

The AFS cell used for migration is called `itsc.austin.ibm.com`.

Figure 13-1 on page 287 shows a snapshot of the AFS file tree to be migrated.

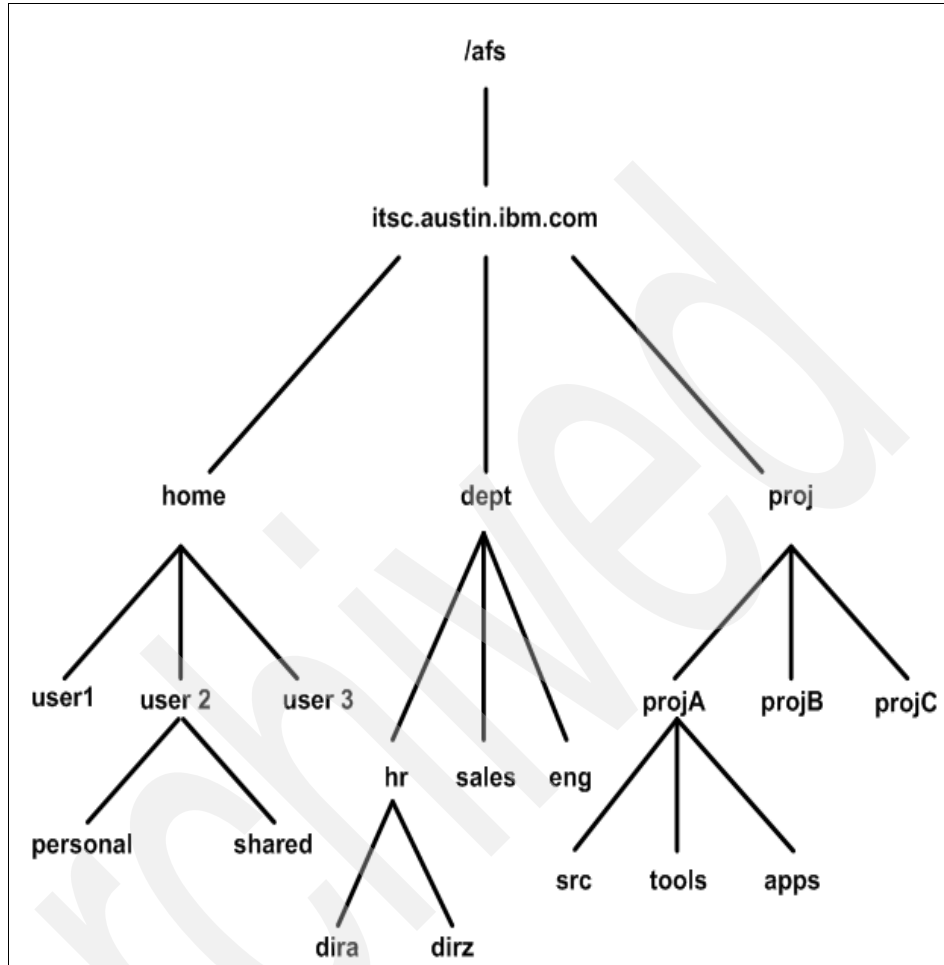


Figure 13-1 Existing AFS Cell namespace

This cell contains users with integrated login enabled on the AFS client machine. Authenticating with AFS is easiest for users if an AFS-modified login utility is installed and configured, because this logs a user into the local file system and obtains an AFS token in one step. A standard UNIX encrypted password was recorded by issuing the standard UNIX **passwd** command. This enables a user to log in to the local file system even after providing an incorrect AFS password. The entries in `/etc/passwd` were later used for getting user data for the migration.

The users are added to various groups. We created volumes and then mounted them at different directories for home areas, projects, departments, and so on. Following a convention of `dira...dirz` and `filea...filez`, the tree was populated with

data. Random ACLs were then set on these directories to simulate an AFS cell environment.

This approach used for the migration can be called one with shared responsibility between users and administrators. The user and group creation, as well as data movement from AFS to NFS, is done by the administrator. The users then set the relevant ACLs for their directories and files.

We provide the exact steps we followed to migrate all this to NFS in the following sections.

13.3 Setting the NFS domain to the AFS cell name

The NFS domain must be set on the target NFS server and client machines. In the sample environment, the NFS domain was set to the AFS cell name. The users who used to see their data at `/afs/itsc.austin.ibm.com/home/user/*` now used the path `/nfs/itsc/home/user/*` on the NFS client machines.

It is mandatory to set the NFS domain name before NFSv4 can be used. NFSv4 requires that the NFS domain be set on all servers and clients. This is because NFSv4 changes the way users and groups are evaluated. Previous versions of NFS used UIDs and GIDs; NFSv4 changed this to `user@domain` and `group@domain`.

The NFS domain name is:

- ▶ By the standard defined by the RFC, bound to the DNS domain name.
- ▶ Case-insensitive; uppercase characters are treated as lowercase during runtime.
- ▶ Uppercase characters are converted automatically to lowercase when using the preferred method to change or set the NFS domain: the **chnfsdom <NFS Domain Name>** command.

If the NFS domain is not set, the output looks as shown in Example 13-1.

Example 13-1 Checking the NFS domain setting with the chnfsdom command

```
# chnfsdom
Current local domain:N/A
#
```

You can check the current setting using the **chnfsdom** or **smitty chnfsdom** command.

Important: We recommend that you do not edit the `/etc/nfs/local_domain` file manually. If this is done and uppercase characters are used, the **chnfsdom** command will display the output exactly as is found in the `/etc/nfs/local_domain` file. This can become confusing because NFSv4 will convert the domain name to lowercase internally. Therefore, great care needs to be taken when editing the file manually.

In the sample environment, the NFS domain is set to achieve the output shown in Example 13-2.

Example 13-2 Setting the NFS domain on the server

```
# chnfsdom itsc.austin.ibm.com
Current local domain:
itsc.austin.ibm.com
```

For simplified management, it is better to logically link the NFS domain name to the DNS domain. In the sample environment, the DNS domain is:

```
itsc.austin.ibm.com
```

Therefore, the NFS domain became:

```
itsc.austin.ibm.com
```

Note: Changing the NFS domain does not recycle or start the **nfsrgyd** daemon. Therefore, the daemon must be manually started or recycled following any changes to the configuration.

The **nfsrgyd** daemon is started using the **startsrc** command.

Example 13-3 shows the full syntax used with the **startsrc** command.

Example 13-3 Starting the nfsrgyd daemon

```
# startsrc -s nfsrgyd
0513-059 The nfsrgyd Subsystem has been started. Subsystem PID is 14496
#
```

Important: The AIX 5L implementation does not require the NFS domain to match your DNS domain. Many styles of NFS domain name can be used, but maintaining a relationship to your DNS domain simplifies managing the environment. It also helps ensure that the name is unique.

13.4 Setting up the KRB5/LDAP environment

We illustrate the various steps to do this in the 6.4, “IBM Tivoli Directory Server V5.2” on page 122.

Note: Check the default stanza in the `/etc/security/user` file and ensure that the `SYSTEM` line looks like the following line:

```
SYSTEM = "KRB5LDAP OR compat"
```

If it is not set to this, modify it before proceeding any further.

13.5 Migrating users to Kerberos and LDAP

Important: Some of the following steps must be performed on the AFS machine, while others are executed on the KRB5/LDAP server. The title for each example states where the step needs to be performed.

Perform the following steps:

1. Obtain admin tokens on the AFS cell and create a directory where all the migration data will reside, as shown in Example 13-4.

Example 13-4 Performing klog on the AFS system

```
# /usr/afs/bin/klog admin
```

```
Password:
```

```
#
```

```
# /usr/afs/bin/tokens
```

```
Tokens held by the Cache Manager (UID Based Tokens):
```

```
User's (AFS ID 1) tokens for afs@itsc.austin.ibm.com [Expires Aug 23 15:27]
```

```
--End of list--
```

```
#
```

```
# mkdir /migration_data
```

2. Create an NFS export on your AFS machine for this migration directory. The LDAP/KRB5 server can then mount this directory and access this data easily.

Example 13-5 Performing mount on the LDAP/KRB5 server

```
# showmount -e istanbul.itsc.austin.ibm.com
export list for istanbul.itsc.austin.ibm.com:
/migration_data (everyone)
#
# mount istanbul:/migration_data /mnt
#
# df -k | grep istanbul
istanbul:/migration_data      540672    429928    21%    1784      1% /mnt
```

3. The next step is to extract user information from the source AFS cell. In 12.3, “Migrating AFS users to NFSv4” on page 269, we discuss different ways of doing this. If it is not necessary to extract the full user information, such as the full names and login shell, you can use the `kas list admin` command. However, this command also lists special AFS users such as *afs* and *admin*. These users should be omitted from the NFS user list, because they are special user accounts that are meaningful only within AFS. We extracted the information from the `/etc/passwd` file. Example 13-6 shows the username, userid, and user description fields being extracted from fields 1, 3, and 5 in this file. This information was written to a file called `user_data.out`. You can also extract the login shell if desired. Home directory settings for existing users can also be extracted, but they are unlikely to map directly to the new environment. This is because most AFS accounts have the home directory set to something in AFS space such as `/afs/itsc.austin.ibm.com/home/user`.

Example 13-6 Performing account extraction on the AFS system

```
# cat /etc/passwd | cut -d: -f1,3,5 > /migration_data/user_data.out
#
# head /migration_data/user_data.out
root:0:
daemon:1:
bin:2:
adm:4:
user1:214:User One
user2:215:User Two
user3:216:User Three
user4:217:User Four
user5:218:User Five
```

4. Next, the `/migration_data/user_data.out` file must be transferred to the LDAP/Kerberos V5 server. But before proceeding any further, it is necessary to ensure that the system accounts (`root`, `daemon`, `adm`) and any user accounts that should not be migrated are removed from the `user_data.out` file.

Note: Note that user IDs and user names are retained from AFS. This will help us later when we add groups and populate the groups with these users. Because we had a fresh KRB5/LDAP installation, we could do this easily. But if you are migrating to a server where there are some existing principals, you need to be careful when adding user and group names to avoid duplication.

5. We used the script **migusr** to populate the KDC and LDAP. Example E-5 on page 368 provides this script. This script creates a special user called *admusr*. This is similar to the AFS *admin* user. It also sets the passwords for all users to a temporary password. Whenever any user logs in to the KRB5/LDAP for the first time, they will have to change their passwords. The user can set it to their AFS password if the user wants to retain the same password. Example 13-7 shows the output of the script.

Example 13-7 Execute migusr script on the KRB5/LDAP server

```
# /mnt/migusr
3004-687 User "admusr" does not exist.
user1:214:User One
user1:214:User One
user1:214:User One
user1:214:User One
Attempting to bind to one or more LDAP servers. This may take a while...
Password for "user1@NFSV4REALM.IBM.COM" changed.
user2:215:User Two
user2:215:User Two
user2:215:User Two
user2:215:User Two
Attempting to bind to one or more LDAP servers. This may take a while...
Password for "user2@NFSV4REALM.IBM.COM" changed.
user3:216:User Three
user3:216:User Three
user3:216:User Three
user3:216:User Three
Attempting to bind to one or more LDAP servers. This may take a while...
Password for "user3@NFSV4REALM.IBM.COM" changed.
```

6. Confirm if the users have been added to KRB5/LDAP. Use the **lsuser** command to list the details about one of the migrated users, as shown in Example 13-8 on page 293.


```
# 1suser -R KRB5LDAP user1
user1 id=214 pgrp=system groups=system home=/home/user1 shell=/usr/bin/ksh
gecos=User One login=true su=true rlogin=true telnet=true daemon=true
admin=true sugroups=ALL admgroups= tpath=nosak ttys=ALL expires=0 auth1=SYSTEM
auth2=NONE umask=22 registry=KRB5LDAP SYSTEM=KRB5LDAP OR compat logintimes=
loginretries=0 pwdwarntime=0 account_locked=false minage=0 maxage=0
maxexpired=-1 minalpha=0 minother=0 mindiff=0 maxrepeats=8 minlen=0
histexpire=0 histsize=0 pwdchecks= dictionlist= fsize=2097151 cpu=-1
data=262144 stack=65536 core=2097151 rss=65536 nofiles=2000 time_last_login=0
time_last_unsuccessful_login=0 unsuccessful_login_count=0 roles=
krb5_principal=user1@NFSV4REALM.IBM.COM
krb5_principal_name=user1@NFSV4REALM.IBM.COM krb5_realm=NFSV4REALM.IBM.COM
maxage=0 expires=0 krb5_last_pwd_change=1125351170 admchk=true
krb5_attributes=requires_preauth,requires_pwchange
krb5_mod_name=admin/admin@NFSV4REALM.IBM.COM krb5_mod_date=1125351170
krb5_kvno=3 krb5_mkvno=0 krb5_max_renewable_life=604800 time_last_login=0
time_last_unsuccessful_login=0 unsuccessful_login_count=0
krb5_names=user1:pecos.itsc.austin.ibm.com
```

13.6 Migrating group information

We now migrate the group information.

Important: Some of the following steps are performed on the AFS machine and some on the KRB5/LDAP server. The title for each example states where the step needs to be performed.

Perform the following steps:

1. First, we have to extract the group information from AFS Protection Database. Use the **pts** command to obtain the pts groups, as shown in Example 13-9. We use the same migration directory created earlier.

Example 13-9 Obtain the group information about the AFS machine

```
# /usr/afs/bin/pts listentries -groups > /migration_data/groups.out
#
# head /migration_data/groups.out
Name      ID      Owner  Creator
system:administrators -204    -204   -204
system:anyuser      -101    -204   -204
system:authuser     -102    -204   -204
staff            -206     1       1
contract         -207     1       1
user11:proja      -208    224     1
friends          -209    229     1
```

Note: We used AFS commands from the /usr/afs/bin directory. On your system, these commands might be inside the /usr/afsws/bin or /usr/afsws/etc directory.

2. AFS user IDs (UIDs) and AFS group IDs (GIDs) have the same function as their counterparts in the UNIX file system, but are used by the AFS servers and the Cache Manager only. Normally, the Protection Server assigns an AFS UID or AFS GID. But we can also assign these values at the time of creation. The AFS UID is a positive integer, and the AFS GID is a negative integer. If we try to retain same GIDs by simply formatting the output from step 1 so that the negative (-) GIDs are converted to positive ones and use those numbers for creating the LDAP groups, there can be a potential problem. The GID that is now a positive number can clash with a UID, and that group will not be created. Therefore, it is best to let the group IDs be generated by the KRB5/LDAP server.
3. Next, we format the output from the command in step 1. We need to replace the owner IDs in text. They will be either user names or other group names. If they are prefixed group names, we need to separate the group name with the owner name and use only the owner name string. We again use /etc/passwd to get the user name relevant to the user ID mentioned. Also, special care needs to be taken for the owner with the UID of 1. This is the AFS admin account. A lot of groups will be owned by this account. Earlier, we created a special user account called admusr to map the AFS admin account. Because we used /etc/passwd to get the user names, UID 1 was replaced with a system account name. Therefore, we need to address this. To accomplish all these steps, we used a script called makereplace. Running this script creates a replace.sed file, as shown in Example 13-10.

Example 13-10 Execute makereplace on the AFS machine

```
# cat /migration_data/makereplace
awk -F":" '{print "s/", $3"/", $1"/g"}{print "s/", $3, "/", $1, "/g"}'
/etc/passwd > replace.sed
awk -F" " '{print "s/", $3, "/", $1, "/g"}' groups.out | grep -v "Owner /
Name">> replace.sed
echo "s/daemon/admusr/g" >> replace.sed
#
# /migration_data/makereplace
```

4. Using the **grpconvert** script, we create another file called new_groups.out. This file has all the data required to migrate groups to the new environment. See Example 13-11 on page 295.

Example 13-11 Execute grpconvert on the AFS machine

```
# cat /migration_data/grpconvert
#!/bin/ksh
cat groups.out | sed -f replace.sed > ./new_groups.out
#
# /migration_data/grpconvert
#
```

5. Check the new groups file to confirm that we are ready to create the groups, as shown in Example 13-12.

Example 13-12 Check the file on the AFS machine

```
# head /migration_data/new_groups.out
Name                ID  Owner Creator
system:administrators -204 system:administrators system:administrators
system:anyuser       -101 system:administrators
system:administrators
system:authuser      -102 system:administrators
system:administrators
staff                -206 admusr admusr
contract             -207 admusr admusr
user11:proja         -208 user11 admusr
friends              -209 user16 admusr
#
```

Note: Ignore the system groups. These are specific to AFS and should not be migrated as is. The NFS administrator will need to carefully give special permissions to users in these groups, because they cannot be mapped directly. At this point, ignore system:administrators, system:authuser, and system:anyuser.

6. Run the **migrate_afs_groups.pl** script to create the groups, as shown in Example 13-13. Example E-6 on page 369 provides this script. The script takes care of both regular and prefixed groups. If it finds a group with the format “owner name:group name,” it extracts the owner and group name. It uses the group name to create the group and sets the owner of that group to the extracted owner name, too. In addition, it makes admusr the owner for all groups.

Example 13-13 Execute the Perl script on the KRB5/LDAP machine

```
# perl /mnt/migrate_afs_groups.pl --file /mnt/new_groups.out
```

7. List the groups created inside the KRB5/LDAP environment, as shown in Example 13-14 on page 296.

Example 13-14 Check the groups on the KRB5/LDAP machine

```
# 1sgroup -R KRB5LDAP ALL
system id=0 admin=true adm id=4 admin=true users=bin,adm registry=KRB5LDAP
uucp id=5 admin=true users=uucp,nuucp registry=KRB5LDAP
.
.
.
contract id=14 admin=true users= registry=KRB5LDAP
proja id=209 admin=false users= adms=admusr,user11 registry=KRB5LDAP
friends id=210 admin=false users= adms=admusr,user16 registry=KRB5LDAP
#
```

8. We now add members to these groups. First, we need to obtain the list of members for each group using **pts** commands, as shown in Example 13-15.

Example 13-15 Use the pts command on the AFS machine

```
# cat /migration_data/migrp
for grpname in `cat groups.out | grep -v ^system | grep -v ^Name | cut -d" " -f1
| cut -d":" -f1`
do
echo \# Exporting group $grpname
for usename in `usr/afs/bin/pts mem $grpname | grep -v "Members of " | grep -v
"is a member of"`
do
echo chuser -R KRB5LDAP groups=$grpname $usename
done
done
#
```

9. Run the **migrp** script to get a file called **add_members**, as shown in Example 13-16.

Example 13-16 Execute the script on the AFS machine

```
# /migration_data/migrp > add_members
#
# head /migration_data/add_members
# Exporting group staff
chuser -R KRB5LDAP groups=staff user2
chuser -R KRB5LDAP groups=staff user3
chuser -R KRB5LDAP groups=staff user4
chuser -R KRB5LDAP groups=staff user5
# Exporting group contract
chuser -R KRB5LDAP groups=contract user6
chuser -R KRB5LDAP groups=contract user7
chuser -R KRB5LDAP groups=contract user8
chuser -R KRB5LDAP groups=contract user9
chuser -R KRB5LDAP groups=contract user10
```

```
# Exporting group user11
# Exporting group friends
chuser -R KRB5LDAP groups=friends user2
chuser -R KRB5LDAP groups=friends user17
chuser -R KRB5LDAP groups=friends user20
chuser -R KRB5LDAP groups=friends user21
#
```

10. Run the **add_members** script on the KRB5/LDAP machine to add the members to each group, as shown in Example 13-17.

Example 13-17 Execute on the KRB5/LDAP machine

```
# /mnt/add_members
```

11. Confirm that the users were added to respective groups in the new environment, as shown in Example 13-18.

Example 13-18 Perform on KRB5/LDAP machine

```
# lsgroup -R KRB5LDAP friends
friends id=211 admin=false users=user2,user17,user20,user21 adms=admusr,user16
registry=KRB5LDAP
#
# lsgroup -R KRB5LDAP staff
staff id=1 admin=false
users=ipsec,ldapdb2,ldap,user40,user50,user60,user80,user90,user3,user4,user5,d
aemon registry=KRB5LDAP
#
# lsgroup -R KRB5LDAP contract
contract id=14 admin=true users=user6,user7,user8,user9,user10
registry=KRB5LDAP
#
# lsgroup -R KRB5LDAP proja
proja id=203 admin=false users= adms=admusr,user11 registry=KRB5LDAP
```

We found it easier to do this in separate scripts so that each step can be easily explained. You can combine some of the previous scripts to have common scripts.

13.7 Migrating data

We chose to migrate a portion of our AFS file tree to NFS. Figure 13-1 on page 287 gives a snapshot of the data that we wanted to migrate. We performed the following steps:

1. We need to create a **tar** for the AFS data that we want to move to NFS. The directory structure on the NFS server has been directly mapped to the existing AFS structure. The home directories of users along with two other directories (proj and dept) were migrated for our sample environment. See Example 13-19.

Tip: If the existing directory structure needs to be changed or hidden by any AFS sites, this is the right place to do this. You might also want to explore the **exname** option.

We use a single NFS server for migrating data. However, in a real-life scenario, you might want to migrate the home, proj, and dept directories on different NFS servers, and then create a namespace using these multiple file systems.

Example 13-19 Tar the data on the AFS machine

```
# pwd
/afs/itsc.austin.ibm.com
#
# tar -cvf /migration_data/afsbackup.tar ./home ./proj ./dept
a ./home
a ./home/user1
a ./home/user1/filea 1 blocks.
a ./home/user1/fileb 1 blocks.
a ./home/user1/filec 1 blocks.
a ./home/user1/filed 1 blocks.
a ./home/user1/dira
.
.
.
a ./dept/eng/fileq 1 blocks.
a ./dept/eng/filer 1 blocks.
a ./dept/eng/files 1 blocks.
a ./dept/eng/filet 1 blocks.
a ./dept/eng/fileu 1 blocks.
a ./dept/eng/filev 1 blocks.
a ./dept/eng/filew 1 blocks.
a ./dept/eng/filex 1 blocks.
a ./dept/eng/filey 1 blocks.
a ./dept/eng/filez 1 blocks.
#
```

Important: The following steps need to be performed on the NFS server machine.

2. On the NFS server machine, create a file system to where the AFS data will be moved, as shown in Example 13-20.

Note: If you want to use NFSv4 ACLs, it is important to select the right type of file system at this time. GPFS and JFS2 with EAV2 are the only ones that support NFSv4 ACLs.

Example 13-20 File system that will be used to hold the migrated AFS data

/dev/fs1v03	131072	130392	1%	4	1% /nfs/itsc
-------------	--------	--------	----	---	--------------

3. Mount the AFS machine so that the /migration_data directory can be accessed by the NFS server machine, as shown in Example 13-21.

Example 13-21 NFS mount from the AFS machine to access the tar files

```
# mount istanbul:/migration_data /mnt
#
```

4. Untar the data onto the NFS server machine, as shown in Example 13-22. The path we used was /nfs/itsc so that the namespaces match.

Example 13-22 Untar the data on the NFS server

```
# cd /nfs/itsc
#
# tar -xvf /mnt/afsbackup.tar
x ./home
x ./home/user1
x ./home/user1/filea, 46 bytes, 1 media blocks.
x ./home/user1/fileb, 46 bytes, 1 media blocks.
x ./home/user1/filec, 46 bytes, 1 media blocks.
x ./home/user1/filed, 46 bytes, 1 media blocks.
.
.
.
x ./dept/eng/filet, 46 bytes, 1 media blocks.
x ./dept/eng/fileu, 46 bytes, 1 media blocks.
x ./dept/eng/filev, 46 bytes, 1 media blocks.
x ./dept/eng/filew, 46 bytes, 1 media blocks.
x ./dept/eng/filex, 46 bytes, 1 media blocks.
x ./dept/eng/filey, 46 bytes, 1 media blocks.
x ./dept/eng/filez, 46 bytes, 1 media blocks.
#
```

5. Check the file tree on the NFS server, as shown in Example 13-23.

Example 13-23 Check if the migrated data exists on the NFS server

```
# cd nfs/itsc
#
# ls
dept      home      proj
```

13.8 Migrating ACLs

The ACL migration is one of the most critical steps because data security and privacy is at stake. The AFS and NFS ACLs are very different in the way of evaluations. We discussed some of these issues earlier in 12.9.3, “Detailed comparison of AFS and NFS ACLs” on page 280. However, we cannot stress this issues enough.

The example that we illustrate is for a single directory.

Note: It is advised that administrators perform the ACL conversion process with caution.

Perform the following steps:

1. If we want to use NFSv4 ACLs, we need to convert the ACL format from AIXC to NFSv4, using the **aclconvert** command, as shown in Example 13-24. The recursive option enables the user to convert ACL types for all the file system objects under a directory structure to the desired ACL type.

Example 13-24 Convert to NFS ACLs on the NFS server

```
# aclconvert -R -t NFS4 /nfs/itsc
```

2. The sample directory used to demonstrate ACL migration is `/afs/itsc.austin.ibm.com/home/user5/dira`, as shown in Example 13-25.

Example 13-25 List the ACLs on the AFS machine

```
$ pwd
/afs/itsc.austin.ibm.com/home/user5/dira
$
$ /usr/afs/bin/fs la .
Access list for . is
Normal rights:
  user11:proja rl
  contract rldw
```

```
staff rl
user5 rlidwka
```

Important: The system:administrators, system:authuser, and system:anyuser groups have been ignored because they hold a special meaning only for AFS. We can create similar groups, such as admgrp, in KRB5/LDAP, but converting ACLs using scripts for groups that hold special meaning can be potentially problematic. The group system:anyuser can potentially be migrated to NFS ACE string - s:(EVERYONE@).

3. We use the **migacl.pl** Perl script to read out the AFS ACLs and convert them to NFS ACLs. Example E-7 on page 370 provides this script. It should by no means be considered a complete solution, but can be used as a basis for a more fully featured tool and customized for a location's particular needs. The prefix path specified in the command line will be used to prefix the path for the destination files on the NFS server. Because we moved our data to /nfs/itsc directory, we used that path. Run this on the AFS machine for a user (for example, user5). Log in to the AFS machine as user5 or set the \$HOME to the home area of user5. The ACL migration script requires this, because it uses the \$HOME environment variable to determine the base path with which it will start. This requires that you remove the AFS /afs/cell_name/ prefix from subsequent **aclput** commands. Also, do not use the short name for the cell. See Example 13-26.

Example 13-26 Run the migacl script on the AFS machine

```
$ perl /migration_data/migacl --prefix /nfs/itsc
Starting at base directory: /afs/itsc.austin.ibm.com/home/user5.
Destination directory: /nfs/itsc/home/user5
Scanning /afs/itsc.austin.ibm.com/home/user5/dira/filea
Scanning /afs/itsc.austin.ibm.com/home/user5/dira/fileb
Scanning /afs/itsc.austin.ibm.com/home/user5/dira/filec
Scanning /afs/itsc.austin.ibm.com/home/user5/dira/filed
Scanning /afs/itsc.austin.ibm.com/home/user5/dira/filee
.
.
$
```

4. The output for this script generates a shell script named **acl_<user_name>.sh** file. In case of user5, this is **acl_user5.sh**. This contains a series of AIX **aclput** commands referencing each file in the **acl_data** directory and its corresponding file. We run this script in step 8. See Example 13-27 on page 302.

Important: The AFS ACLs are at a directory level, and NFS ACLs are at a file level. Therefore, when migrating ACLs, we need to set the NFS ACL on each and every file inside a directory.

Example 13-27 Perform on the AFS machine

```
$ pwd
/afs/itsc.austin.ibm.com/home/user5/dira
$
$ ls
acl_user5.sh  filee      filej      fileo      filet      filey
filea         filef      filek      filep      fileu      filez
fileb         fileg      filel      fileq      filev
filec         fileh      filem      filer      filew
filed         filei      filen      files      filex
$
$ cat acl_user5.sh
#!/bin/sh
aclput -i /nfs/itsc/home/user5/acl_data/filea.acl
/nfs/itsc/home/user5/dira/filea
aclput -i /nfs/itsc/home/user5/acl_data/fileb.acl
/nfs/itsc/home/user5/dira/fileb
aclput -i /nfs/itsc/home/user5/acl_data/filec.acl
/nfs/itsc/home/user5/dira/filec
aclput -i /nfs/itsc/home/user5/acl_data/filed.acl
/nfs/itsc/home/user5/dira/filed
aclput -i /nfs/itsc/home/user5/acl_data/filee.acl
/nfs/itsc/home/user5/dira/filee
```

5. If you go one level up, we see an `acl_data` directory, as shown in Example 13-28. This contains NFSv4-style `.acl` input files for each file.

Example 13-28 Perform on the AFS machine to see the files generated from script

```
$ cd ..
$
$ ls
acl_data  dire      dirj      filea     filef     filek     filep     fileu
filez
dira      dirf      dirk      fileb     fileg     filel     fileq     filev
dirb      dirg      dirl      filec     fileh     filem     filer     filew
dirc      dirh      dirm      filed     filei     filen     files     filex
dird      diri      dirn      filee     filej     fileo     filet     filey
$
$ cd acl_data
$
$ ls
```

acl_user5.sh.acl	filef.acl	filel.acl	filer.acl
filex.acl			
filea.acl	fileg.acl	filem.acl	files.acl
filey.acl			
fileb.acl	fileh.acl	filen.acl	filet.acl
filez.acl			
filec.acl	filei.acl	fileo.acl	fileu.acl
filed.acl	filej.acl	filep.acl	filev.acl
filee.acl	filek.acl	fileq.acl	filew.acl

6. An ACL input file constructed using this Perl script, as found in the `acl_data` directory, might look similar to the sample in Example 13-29. This is a rough approximation of the ACL found on the matching file located in the AFS namespace. The `<filename>.acl` input file are generated by the Perl script for a file called `filea`. You can compare it to the AFS ACL in step 2.

Example 13-29 Sample of a converted ACL

```
$ cat filea.acl
*
* ACL_type   NFS4
*
*
* Owner: user5
* Group: staff
*
u:user5:    a      racxwpdDwC
g:staff:    a      racx
g:contract: a      racxwpdDw
g:proja:    a      racx
s:(OWNER@): a      racxwpdDwC
$
```

7. This ACL data needs to be moved to the NFS server where the script to set these ACLs will be executed, as shown in Example 13-30.

Example 13-30 Move this data to the NFS server

```
$ pwd
/afs/itsc.austin.ibm.com/home/user5/dira
$
$ mkdir /migration_data/user5
$
$ mv /afs/itsc.austin.ibm.com/home/user5/acl_data /migration_data/user5
$
$ cd /afs/itsc.austin.ibm.com/home/user5/dira

$ cp acl_user5.sh /migration_data/user5
$
```

8. The **acl_user5.sh** script needs to be executed on the NFS server machine. Login to `pecos.itsc.asutin.ibm.com` as `user5`. Because this user is logging in to KRB5/LDAP for the first time, the temporary password set while creating users needs to be used. We used it in our **migusr** script. A password reset will be enforced, and the user can reset the password to the old AFS password if desired. After the user logs in, the script for setting ACLs on the NFS server needs to be run. Remember to check that `user5` owns these files. See Example 13-31.

Example 13-31 Perform on the NFS server

```
$ cd /mnt/user5
$
$ cp -r /mnt/user5/acl_data /nfs/itsc/home/user5/
$
$ ./acl_user5.sh
$
```

9. Check the ACLs on the NFS server, as shown in Example 13-32.

Example 13-32 Check the ACLs on the NFS server

```
$ pwd
/nfs/itsc/home/user5/dira
$
$ aclget filea
*
* ACL_type   NFS4
*
*
* Owner: user5
* Group: system
*
u:user5:      a      rwpxDadcC
g:staff:      a      rxac
g:contract:   a      rwpxDadc
g:proja:      a      rxac
s:(OWNER@):   a      rwpxDadcC
$
```

Similarly, other ACLs can also be converted. Users might opt to delete the `acl_data` directory and the `acl_<username>.sh` file itself, because they will no longer be needed after the AFS to NFSv4 ACL conversion process has been completed.

13.9 Accessing the migrated data from NFSv4 clients

Now, we use the NFS RPCSEC_GSS protection mechanism to access the migrated data from NFS. We use the Kerberos V5, KRB5, security option.

1. Check if the NFS client is added to the NFS domain, as shown in Example 13-33.

Example 13-33 Perform on the NFS client

```
# chnfsdom
Current local domain: itsc.austin.ibm.com
#
```

2. Because we are going to use the KRB5 security feature, we need to check the status of the **gssd** daemon on both the server and client machines, as shown in Example 13-34. But first, we need to configure a secure NFS environment. In 9.6, “Adding security” on page 206, we provide the steps you must complete before using these NFS features.

Example 13-34 Check if the gssd daemon is started

```
# lssrc -s gssd
Subsystem      Group      PID      Status
gssd           nfs       520438   active
#
```

3. Export the data on the NFS server pecos.itsc.austin.ibm.com, as shown in Example 13-35.

Example 13-35 Export the data on the pecos server

```
# cat /etc/exports
/nfs/itsc -vers=4,sec=krb5,rw
#
# exportfs -va
Exported /nfs/itsc
#
```

4. Mount this exported data from the NFS client, sabine.itsc.austin.ibm.com, as shown in Example 13-36.

Example 13-36 Mount this on the sabine client

```
# mount -o vers=4,sec=krb5 pecos.itsc.austin.ibm.com:/ /mnt
```

5. Try accessing the migrated AFS data from this NFS client. Log in as a user and create some new files in your home directory to test whether you have the necessary permissions. If the ACLs have been migrated properly, you will be able to do so. See Example 13-37.

Example 13-37 Creating a file with correct ACLs set

```
$ cd /mnt/nfs/itsc/home/user1
$
$ ls
dira  dire  diri  dirm  filec  fileg  filek  fileo  files  filew
dirb  dirf  dirj  dirn  filed  fileh  filel  filep  filet  filex
dirc  dirg  dirk  filea  filee  filei  filem  fileq  fileu  filey
dird  dirh  dirl  fileb  filef  filej  filen  filer  filev  filez
$
$ touch test_file
$
```

6. Now, try to create files in a directory where you did not have permissions while they were in AFS, as shown in Example 13-38. You should continue to fail here, too. This is the way that we can test the ACL migration.

Example 13-38 Attempt to create a file when the ACLs do now allow the operation

```
$ cd /mnt/nfs/itsc/dept/hr/dira
$
$ touch test_file
touch: 0652-046 Cannot create test_file.
$
$ aclget .
*
* ACL_type  NFS4
*
*
* Owner: daemon
* Group: system
*
s:(OWNER@):  a      rwpRWxDaAdcCs
s:(OWNER@):  d      o
s:(GROUP@):  a      rRxadcS
s:(GROUP@):  d      wpWDACo
s:(EVERYONE@): a      rRxadcS
s:(EVERYONE@): d      wpWDACo
$
```

Notice that NFS ACL `s:(EVERYONE@)` does not have `w` permissions in this directory.



Part 5

Appendixes

This part contains the appendixes, providing information about our test system, a case study, and how to configure the Network Time Service. We also provide a quick reference, scripts and configuration files, information about how to install an AIX maintenance level, a planning worksheet, and information about the additional material provided with this book.

Test environment

This appendix diagrams the environment that was used to model the examples provided in this book.

Table A-1 on page 310 lists the systems and Figure A-1 on page 311 shows the layout and names of the systems used to create the environments used in this book.

In the test environment used to illustrate examples in this book, we found it useful to exploit the abilities of the POWER5 architecture with AIX 5L. By using logical partitions on the POWER5 platform, we created NFS servers and clients to test various migration strategies. Although the test environment was relatively simple compared to a production migration, the concept of using LPAR and Micro-Partitioning server technology can significantly reduce migration costs from both a test environment and production environment standpoint. This technology allowed for a much larger test environment than the physical number of machines that were used.

Table A-1 Systems used to create the test environment for this book

System name	OS version	Function	Additional functions
pecos	AIX 5L V5.3 RML01	KRB5/LDAP server	KRB5: IBM Network Authentication Services LDAP: IBM Tivoli Directory Server
guadalupe	AIX 5L V5.3 RML03	KRB5 client	
sabine	AIX 5L V5.3 RML03	KRB5 client	GPFS node
frio	AIX 5L V5.3 RML03	KRB5 client	GPFS node
angelina	AIX 5L V5.3 RML03	KRB5 client	GPFS node
brazos	AIX 5L V5.3 RML03	KRB5 client	
trinity	AIX 5L V5.3 RML03	KRB5 client	
madrid	AIX 5L V5.3 RML03	NIM server	
istanbul	AIX 5L V5.2 RML04	AFS sever	Source for AFS migration scenario
milan	AIX 5L V5.2 RML04	DFS server	Source for the DFS migration scenario

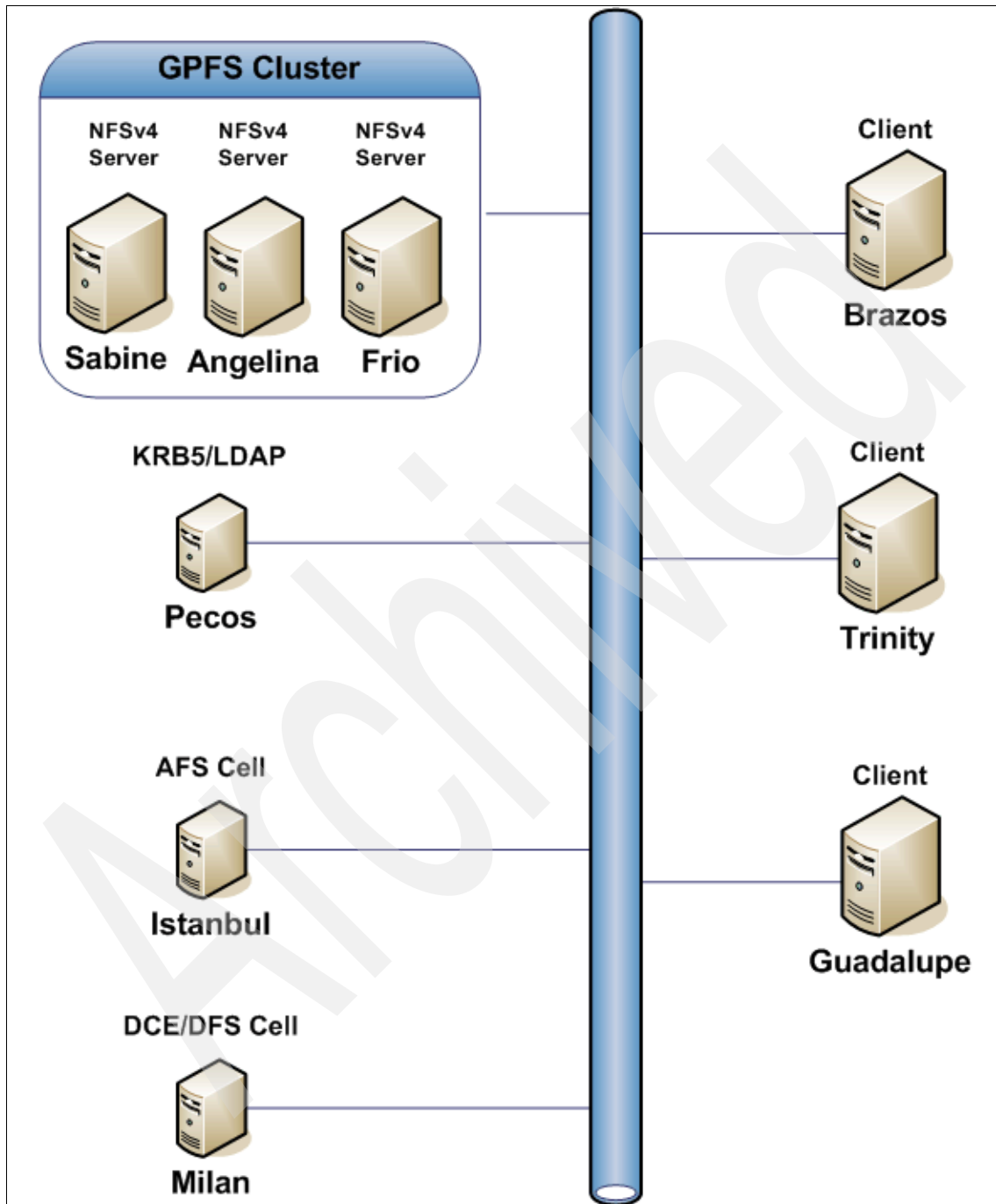


Figure A-1 Environment used to create the examples in this book

Case study: IBM Global Storage Architecture

This appendix provides an overview of a solution that was architected for IBM internal use by the IBM CIO, IBM Global Services, and IBM internal customers. The Global Storage Architecture (GSA) File service was developed to replace AFS and DFS as part of the new development infrastructure of IBM.

We discuss some of the business problems addressed, project approaches, and details of the solution architecture.

Business problem

During the 1990s, the majority of the IBM hardware and software development labs deployed two versions of enterprise file systems: AFS and DFS. By the end of 2000, more than 100 TB of data was stored in these file systems worldwide.

At that time, it became clear neither AFS nor DFS was a good long-term solution due to an approaching end of life status for each product. The IBM CIO wanted to deploy a replacement for these file systems, as well as a mix of IBM OS/2® and Microsoft Windows based offerings, into a single, worldwide offering that would offer improved service at a reduced cost to the IBM account. This new system would be called the Global Storage Architecture, or GSA.

The GSA File project was intended to replace all current enterprise file systems deployments within IBM, including AFS, DFS, NFS, and Common Internet File System (CIFS). This meant that the design had a wide range of customers to satisfy, from the high performance demands of the current AFS/DFS users, to the low-end casual users that need a simple file repository.

A challenge was faced in integrating the various architecture components that were selected, not only from a design aspect, but also in consideration of the existing IBM infrastructure. It was impossible and undesirable to attempt an overnight migration from the existing systems to GSA File. The new system would have to interoperate with the existing systems for a period of years during the migration.

The technology was changing rapidly during the design process. Certain IBM products (such as the IBM RS/6000 Scalable Parallel system) were selected that, in some cases, did not have a clear future growth path. Some offerings in development within IBM (such as Storage Tank™, later announced as the IBM SAN File System) looked promising, but these products were not yet sufficiently stable to consider deployment in a production environment at the time. This forced decisions regarding the design, allowing for the integration of such in-development technologies at some future time, without affecting the end users of the service.

In some cases, considerable customer resistance to change was encountered. Many customers had integrated the existing file systems into their various development environments and did not want or see a need to migrate to another solution. This led to some requirements that would have simply been impossible to meet.

It was necessary to examine all the requirements in order to match each one against the technology that was available. This had to be done in a manner that permitted the integration of future technology without major disruption and without requiring another migration to a future file system.

Solution

The first task was to identify the requirements for the new file systems offering. Through a series of meetings with key internal customers, the team identified these key requirements:

- ▶ Longevity
- ▶ Ability to handle many large and small files
- ▶ Caching
- ▶ Excellent reliability, availability, and serviceability
- ▶ “Usable” ACLs
- ▶ Large numbers of groups
- ▶ Location transparency
- ▶ Quotas, allocation method, and control of growth
- ▶ Common authentication
- ▶ Billing by usage
- ▶ Reduced cost of administration
- ▶ Scalability
- ▶ Consistency and a common toolset
- ▶ Ability to archive data to safe site
- ▶ 24x7 availability
- ▶ Perform better than the existing file systems
- ▶ Backups performed daily
- ▶ Ability of users to restore their own data from backups or snapshots

Using those basic requirements, a study determined what pieces of technology could be used to satisfy each request. A number of different systems were evaluated that were structured similarly to AFS. While several of them had interesting feature sets, none were sufficiently stable for production use. NAS type solutions were also evaluated, but it was necessary to avoid a situation where we had a fragmented namespace and thousands of NFS mount points.

Some cluster file systems looked interesting, but loading proprietary clients on every workstation represented an undesirable solution. This would be especially difficult because of the wide variety of systems in use within IBM, each with different management structures and varying levels of currency. The environment targeted for support included everything from software development shops with

multiple and varying levels of operating systems, to hardware design shops with thousands of machines that are identically configured and centrally managed.

In the end, a hybrid design was selected. A clustered file system was chosen that provided a large tree with a consistent namespace, but which exported it using standard protocols such as CIFS and NFS. It was felt that this provided the best of both worlds: a large, easily managed pool of space with support for industry-standard clients.

Each GSA File cell has at its heart a GPFS cluster. Initially, each site was referred to as a GSA File cluster, but the cell terminology from AFS and DFS kept returning. Early on it was determined that the “cell” terminology was too ingrained into both the development team and its customers, so the terminology was changed, and they were redesignated as GSA File cells.

Figure B-1 on page 317 shows the GSA File components.

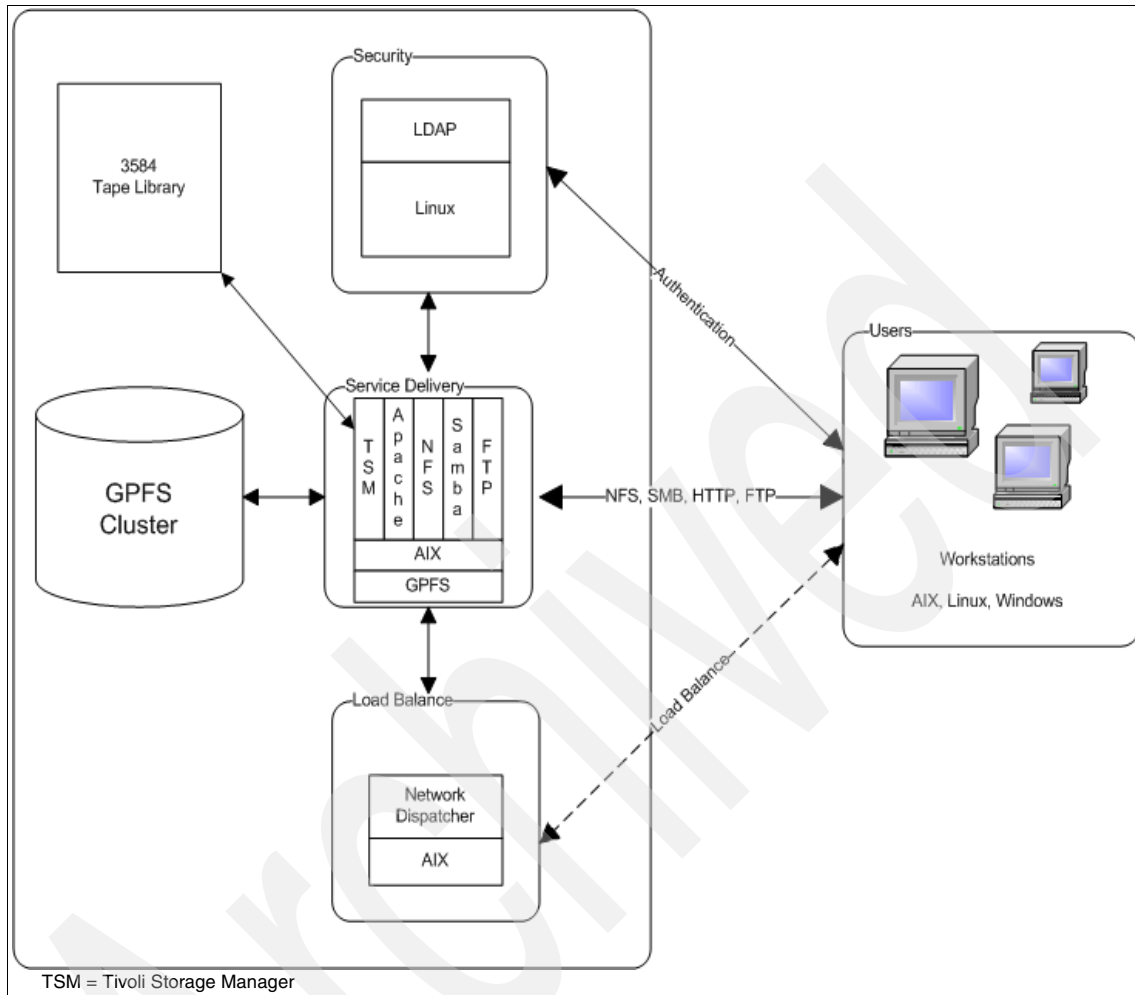


Figure B-1 GSA File components

GPFS File system

A scalable, robust back-end file system was required to support GSA File. The General Parallel File System (GPFS) was determined to meet all established requirements. GPFS file systems had been widely used on the IBM RS/6000 Scalable Parallel (SP) system platform for several years, so they had a proven track record. GPFS met many of the solution's requirements, allowing storage and nodes to be added and removed dynamically. It also supported read/write replication, a highly desirable feature found in both AFS and DFS.

GPFS is a shared device file system in that every node in a GPFS cluster has full read/write access to the disks in a file system. GPFS coordinates this access using a distributed locking mechanism in order to maintain consistency on disk. The GPFS locking mechanism relies on having a quorum of servers available. The full GSA File cell uses a minimum of three servers in the GPFS cluster, so quorum is maintained if one server fails.

This shared device design allows the file system to grow very large, and a large number of nodes can participate in a GPFS cluster with very large aggregate disk bandwidth. The shared device aspect of GPFS also entails unique security concerns. Because every node can read and write every disk, client systems could not be permitted to participate directly in GPFS. Only server nodes controlled by GSA File administrators are allowed to participate in GPFS. All client systems access the file system through network protocols such as CIFS and NFS.

When GPFS first became available, it operated only over the SP high-speed switch, on a subsystem called VSD for Virtual Shared Device. GPFS was a SAN file system, but in the days before SAN technology became common, it effectively ran on a virtual SAN. As the GSA File project began, GPFS began supporting SAN-attached Fibre Channel disks, and also began running on AIX systems that were not part of a SP cluster.

GPFS supports a wide variety of block sizes, approaching the megabyte range. It was found that smaller block sizes provided better performance for CIFS and NFS clients because they matched more closely the read and write sizes used by the clients. All GSA File cells are configured with a 64 K block size in GPFS. GPFS divides each block into 32 subblocks for allocating space. With 64 K blocks, the sub blocks are 2 K. This is the minimum amount of quota that can be allocated for small files, and it has worked well because it matches the expectations of users moving from the existing systems.

Security

During development, different systems were evaluated for providing a user and group security infrastructure. Based on both internal and industry trends, LDAP was the obvious choice. We based the GSA File schema on the industry-standard schema defined in RFC 2307, adding a few object classes and attributes as needed to manage individual requirements for the users and infrastructure.

Availability and performance are critical for a security system deployed in support of a large enterprise file system. The decision was made to deploy replicas of the LDAP directory at every GSA File site; the large number of replicas dictated a tiered replication strategy. One master LDAP server and a hot spare were

deployed in Poughkeepsie NY, with a submaster at each GSA File site. Two geographically dispersed submasters were located in Europe and Asia. The master accepts all changes to the directory and pushes them to the geography submasters as well as the North American site submasters. The site submitters then push changes to the end-user replicas at each site.

The master server is a single point of failure for the system, but only for changes to the LDAP directory. If a user at a specific cell cannot reach the master on the network, or the master is down, that user can continue to use the file system with data located on their local replicas but cannot change their password, manage groups, or make other changes to the directory. This was considered a good compromise between creating locally managed users and groups at each site and having a central user ID and group database.

Load balancing

Both hardware-based load balancers and the software-based IBM Network Dispatcher were considered for this role. The Network Dispatcher platform is part of IBM WebSphere Edge Server, and it runs on a variety of platforms.

The flexibility of running a load balancer in software on a server was a tremendous benefit, especially because the load balancer products were focused on Web protocols, and GSA required balancing of file system protocols that were not typically load balanced.

Network Dispatcher is designed so that only inbound packets to the servers must flow through the load balancer. Outbound packets from the servers to the client go directly across the network. This permits the use of fewer load balancers than might otherwise have been required in an environment such as GSA, where reads typically outnumber writes.

Each GSA File cell starts with a single Network Dispatcher cluster, which is a publicly advertised IP address. All client traffic comes to the cell over that IP address. Over time, as the workload grows, additional clusters are put into service. This allows more network dispatchers to be added and provides more throughput. The additional clusters are added to a round-robin DNS entry so that clients are randomly assigned to a cluster. The load is actively balanced across the back-end systems within each dispatcher cluster.

The dispatcher systems themselves are arranged in pairs using the dispatcher high availability function. This way, if a dispatcher fails, another will take over its cluster and service its clients with degraded performance. By keeping all of the clusters available at all times, there is no need to change DNS records or rely on DNS timeouts to maintain availability for the cell.

Figure B-2 illustrates the high-level architecture of the GSA File.

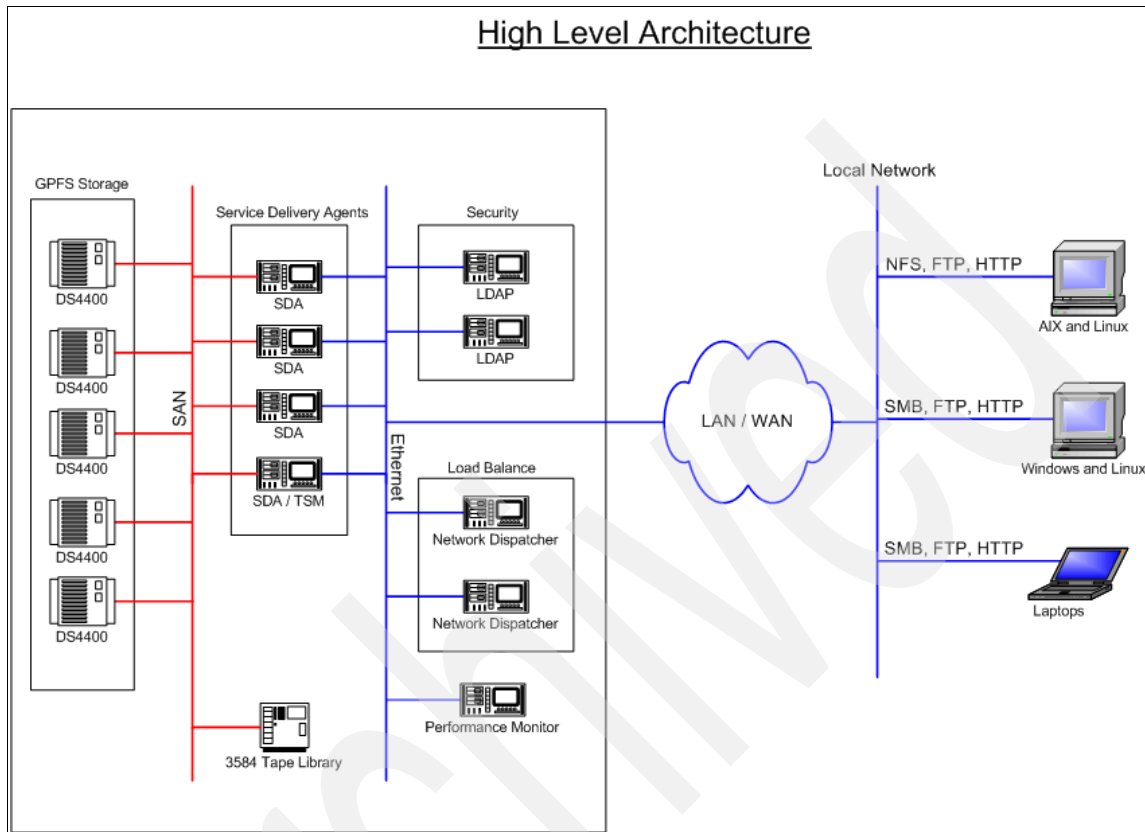


Figure B-2 GSA File high-level architecture

Server hardware

There are three main classes of servers in the GSA File environment: file servers, directory servers, and dispatchers.

The initial selection for file server hardware platform was effectively determined by the available hardware. When GPFS deployment began, it only ran on AIX in the SP environment, which determined hardware deployed in the first GSA File cells. Quickly thereafter, GPFS was supported on IBM @server pSeries systems, and clustered pSeries servers began to be deployed instead. As noted earlier, each full GSA File cell starts with three file server nodes so that GPFS can maintain a quorum if one fails.

Since that time, GPFS has been ported to Linux on IBM @server xSeries®, but current plans involve it also remaining in use on AIX 5L and the pSeries platform in GSA File. Experience has shown that the NFS server on AIX 5L is more robust than the NFS server on Linux.

IBM was searching for opportunities to deploy Linux infrastructure at the time the GSA File project began, and it was felt the directory service in GSA File would be a perfect fit. Each GSA File cell starts with three Linux directory servers, one submaster and two replicas.

Storage

The use of a shared device file system such as GPFS led to Fibre Channel-attached storage. But at the same time, this was not a typical SAN deployment. There was no need to segment the SAN or the storage servers to provide different sets of disks to different servers. Every node on the SAN would be able to read and write every disk. As a result, the management tools in many of the high-end disk subsystems were not required.

It was understood GPFS could provide a high level of reliability. Because GPFS would be replicating all data onto two different storage servers, slightly lower reliability from each of the storage servers was an acceptable compromise.

These considerations led to the decision to reduce costs by deploying the IBM midrange disk offering, then called FASTT and now called the IBM TotalStorage® DS4000 series. The midrange storage systems provided performance and availability that was good enough, at a lower cost than the high-end systems. They also permitted growth to the system in smaller increments, which was very important during the initial rollout phase.

RAID 5 arrays are created on each midrange storage server, and each array is exported as a single LUN. The LUNs are grouped by GPFS into failure groups, which are groups of LUNs that share common failure modes. Typically, only two failure groups are configured per GSA File cell, and all of the LUNs from a given storage server are configured into one of the failure groups. GPFS creates two copies of every data and metadata block, with one in each failure group. This allows the file system to be kept online even if an entire storage server is lost.

Two mostly independent switch fabrics are also maintained between the file servers and the storage servers. This allows the file system to remain online even if a switch or link is lost.

Protocols and software

The CIFS and NFS protocols are the primary means of accessing GSA File. Certain “lightweight” protocols are also provided for access to the system; these include HTTP, FTP, SCP SFTP, and rsync over SSH. Finally, support is offered for auxiliary protocols that provide file system access: NetBIOS Name Service (NBNS), also known as WINS, and NTP. All these tasks are accomplished using a mixture of IBM products and open source software:

- ▶ The CIFS protocol is supported in GSA File by Samba running on AIX 5L systems. Several different CIFS solutions were evaluated, and it was found that Samba was both robust and flexible. The same protocol is used to provide an NBNS or WINS service that is proxied to DNS.
- ▶ An AIX 5L built-in NFS server subsystem is used to provide NFS service to clients. It has proven to be very robust in the GSA environment.
- ▶ Apache provides HTTP access, both to the file system and to the GSA management tool suite.
- ▶ After evaluating a number of different FTP servers, ProFTPD was selected for GSA File. ProFTPD supports modules, one of which integrates well with an RFC 2307 LDAP service. It also handles chroot FTP service well, because it does not require any external programs such as `ls` to provide this service. Although authenticated FTP service is available, users are encouraged to use `sftp` instead because it encrypts their passwords on the wire.
- ▶ Finally, several secure means are available for accessing the file system through OpenSSH and the rsh restricted shell. The rsh shell enables users to be restricted to file transfer and rsync operations and can **chroot** them into the GSA File file systems so that they cannot access local file systems, such as `/tmp`, on the servers.

All these file access protocols are served from every file server in GSA File. This helps keep management simple by keeping the file servers alike. It also helps to keep the workload evenly distributed and improves availability over a system where individual protocols had dedicated servers.

Backups

All customer data in GSA File is backed up using IBM Tivoli Storage Manager. The Tivoli Storage Manager server and client services are available on a file server node in each GSA File cell. The dispatchers are configured not to send end users to this node under normal circumstances, but it provides the full complement of end-user services in case it is needed.

The IBM Linear Tape-Open (LTO) family tape libraries used by Tivoli Storage Manager are attached to the GSA File SAN. This enables LAN free backups,

because the node running Tivoli Storage Manager reads the data from the SAN-attached disks and writes it back to tape on the SAN.

Two copy pools of the data are kept on site to protect against tape failures. An additional copy is sent off site to safe storage.

Time synchronization

Time synchronization is extremely important within any enterprise file system. The time stamps on files and directories are updated by file servers in certain instances, and directly by clients in others. This can lead to problems if the clocks are not synchronized across the system. Systems that use Kerberos must also have clocks synchronized within a certain window.

Technically speaking, it is not important whether the clocks in the system are accurate, so long as they are synchronized with each other. Within GSA File, the decision was made to synchronize with an accurate and stable time source, resulting in selection of a global positioning system (GPS) as a time provider.

Three Stratum One time servers with GPS receivers were deployed at three different sites in North America. All directory servers in the GSA File infrastructure synchronize their clocks with these servers using Network Time Protocol (NTP). All remaining servers in GSA File synchronize their clocks with the three (or more) directory servers at their site.

GSA File clients synchronize their clocks using NTP with the cluster address for their local cell. The NTP packets that arrive at the network dispatchers are forwarded to a file server, which services NTP requests from clients.

Kerberos and NFSv4

The GSA File project began using NFS version 3, but NFS version 4 was planned for long before it became commercially available. NFSv4 provides the strong security that was partially lost during the migration away from AFS and DFS. Kerberos V5 is now being rolled out on all GSA File directory servers. Deployment of NFSv4 in production will commence after the Kerberos deployment is complete. The Kerberos support in NFSv4 is probably the single most compelling feature for GSA File.

Another key feature of NFSv4 for GSA File is file delegation. A strong need exists for improving the ability of NFS clients to cache data locally.

Some other NFSv4 features related to namespace, such as referrals and replication, are interesting for use with GSA, but are not as critical in the

clustered environment. These will be examined over the long term to see how they can be used to benefit GSA File clients.

Centralization

The AFS and DFS file systems inside IBM grew over a period of time in a grass roots fashion. This meant that every site had its own implementation, tools, and policies around AFS and DFS. While this provided tremendous flexibility, it also contributed to increased cost and often led to problems being rediscovered and solved independently at multiple sites. It also led to customer frustration as more projects became cross-site projects, resulting in administrators being forced to deal with multiple systems that were managed differently.

In contrast, GSA File is a centrally managed, lights-out system. This leads to lower cost and greater consistency than the existing systems. It also means that when a problem has been found and solved in one cell, the solution is immediately available at all other cells. The management structure in GSA File is based around a control center team located in Poughkeepsie, NY. They manage the master directory server and all GSA File cells in North America. They are supported by a subsidiary control center teams in Europe and one in Asia.

Outside the three control center sites, each cell requires only occasional on-site support by personnel, who perform such duties on an as-needed basis. Most issues and management functions are resolved over the network by the control center team. In the event that a disk or other hardware component fails, the control center team pages a local support contact to resolve the issue.

The net result is that GSA File service is consistent worldwide, in terms of service levels, tools, and policies. The cost for space is also significantly lower than that incurred by the existing environments.

Scalability

The GSA File cells described to this point have been rather large. They include at least three file servers, two storage servers, two SAN switches, three directory servers, and two dispatchers. They are fully redundant and thus can tolerate the failure of any piece of hardware. They can scale to be extremely large and serve tens of thousands or more client systems.

These cells are not such a good fit for small, remote sites. This type of site generally requires a small amount of storage and would benefit from the same tool set as the large sites, but has no requirement for access to terabytes of storage. Such sites' network capacity is also frequently insufficient or incapable of providing low enough latency to use a remote cell. A fully redundant cell would not be cost effective for storage volumes below one terabyte.

For such small sites, the option exists to deploy what is called a GSA File Lite cell. This type of cell includes a single file server and a single directory server with no separate dispatcher. Mirrored storage is included, but no redundancy of server hardware. This configuration allows remote development teams to use the same type of storage with the same tools that larger teams use in a cost-effective way on a small scale.

GSA File Lite cells trade cost efficiency at small volumes for reduced availability. It has proven to be a compromise that works well for GSA File cells in places such as Israel and South Africa.

Benefits of GSA File

GSA File provides a number of benefits over the AFS and DFS environments it replaces.

The primary benefit is that the file servers in GSA File are virtualized. Every file server has access to all the customer data within a GSA File cell. Client systems are directed to the file server with the lightest load, rather than to the server that houses the requested data. This means that the system can continue to function even if a file server fails. It also allows file servers to be added and removed to handle changing loads without interrupting the service.

In AFS and DFS environments, file server load balancing is accomplished by moving data from server to server. Moving volumes and filesets takes time and resources, especially for busy data. And while the process can be automated, it is always reactive. The clustered GPFS file system and the dynamic load balancing of the network dispatchers in GSA File means that the file servers remain balanced without administrator intervention.

The virtualization and dynamic load balancing also offer a tremendous performance benefit over traditional file systems under heavy load. In a traditional file system, where every file server owns a particular part of the tree of read/write data, hot spots in the data translate directly to hot spots on the servers and disks. In GSA File, all servers can access all the data, so if thousands of clients attempt to read a file, the load will be spread across all of the file servers. GPFS also stripes the data itself across all of the disks. Where a traditional file system must have enough surplus capacity on every file server to handle peak loads and spikes, the GSA File system spreads such loads across multiple servers and can keep a smaller total amount of surplus capacity online. Simply put, GSA File cells can handle larger peaks with less hardware than traditional file systems.

The use of industry-standard protocols such as CIFS and NFS provide tremendous benefits to GSA File because GSA File uses commodity clients. The AFS and DFS file system clients limited the number of supported operating systems. They also often added a six-month delay after an operating system release before it could be used with the file system. A wider variety of clients can be supported on GSA File, with fewer problems, because the file system client software comes with the operating system. This is particularly beneficial for groups that develop software because they can use prerelease and early-release operating system versions with GSA File.

The clustered design of GSA File provides many of the benefits of the proprietary AFS and DFS clients using commodity clients. In a sense, many of the functions of the AFS and DFS clients, such as namespace management, have been pulled back into the data center with GSA File. In AFS and DFS, the clients require enough intelligence to locate the server holding the data for which they are looking. In GSA File, the clients access what looks like a single server per cell, and the servers in the data center locate the data they need.

GSA File status

The GSA File project is progressing well and exceeding expectations. There are 21 GSA File cells in production today on five continents. There are more than 75 thousand GSA File users with more than 70 terabytes of data. The monthly growth rate as migration progresses away from AFS and DFS has remained at approximately 5%.

Figure B-3 on page 327 shows the GSA File cell locations.

GSA Cell locations

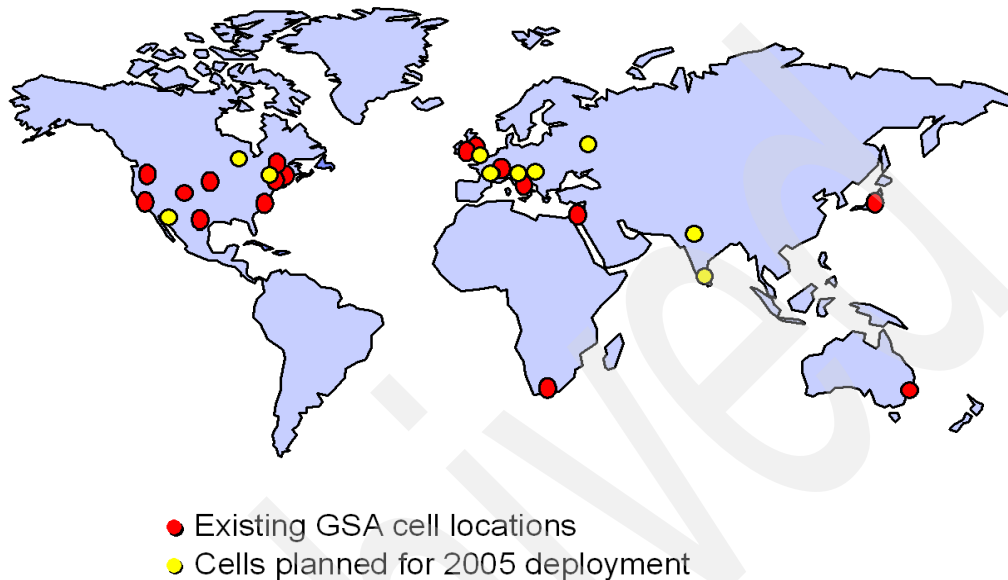


Figure B-3 GSA File cell locations

The GSA File cells have demonstrated significantly higher availability than the AFS and DFS systems. GSA File availability is measured 24x7 on a system basis, with outages recorded whenever any GSA File protocol is down in a cell. Even with this strict standard, GSA File cells are exceeding the availability of the AFS and DFS cells inside IBM, which are often measured on a 5x10 basis using the average server availability as a metric. This means that in AFS and DFS cells, if one server out of 60 is down for an hour, only one minute of outage is recorded for the cell.

The cost of GSA File space has also remained lower, and is dropping more quickly than the cost of AFS and DFS inside IBM. In file service, as in almost any IT service, the largest cost component is labor, and the GSA File project has focused strongly on reducing costs in this area. Two major factors contribute to the cost difference. The first is centralization to reduce duplicate functions. The second is that space in GSA File is managed in larger pools than in AFS and DFS. A large GSA File cell might have four file systems instead of hundreds of aggregates and tens of thousands of filesets or volumes. With fewer places to run out of space, less administrator time is required to manage it.

GSA File is an evolving system. Changes are constantly being made in order to keep up with technology, reduce costs, and improve service. The biggest change so far is the deployment of NFS version 4. It is expected that the rate of migration from AFS and DFS will quicken dramatically due to the improved security and caching available in the most recent NFSv4 release.

Configuring Network Time Service

This appendix describes some example configurations for the Network Time Service (NTP) on AIX 5L.

In order to use NTP, at least one system in your environment must be configured as an NTP server. This system can either act as an independent time source (no reference clock) or can be configured to reference a precision time source (that receives time from the U.S. global positioning system or another source). We do not provide the details about how to install and configure this reference time server in this appendix.

Configuring the NTP server with a reference clock

Figure C-1 shows a sample environment with clients getting their time from a local time server on your network. The local time server, in turn, gets its time from an external reference clock.

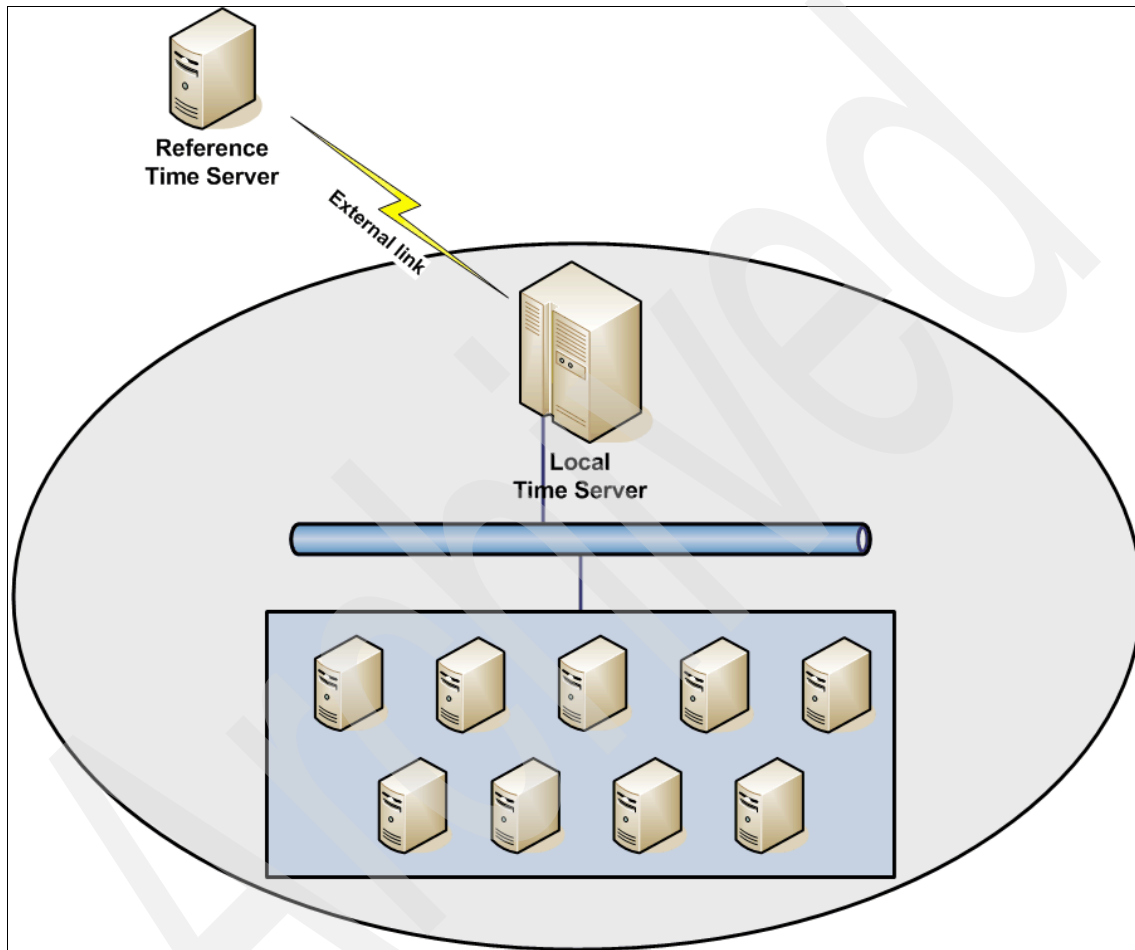


Figure C-1 Configuring XNTP with an external reference clock

Perform the following steps to configure the NTP server with a reference clock:

1. Make sure that the xntp subsystem is not running. If it is, stop it. Example C-1 on page 331 shows the full sequence of commands.

Example: C-1 Stopping the xntp subsystem on the server

```
# lssrc -s xntpd
Subsystem      Group      PID      Status
xntpd          tcpip      479410    active
#
# stopsrc -s xntpd
0513-044 The /usr/sbin/xntpd Subsystem was requested to stop.
```

If the subsystem was not running, the first command in Example C-2 would show the status of the daemon as inoperative.

2. Edit the /etc/ntp.conf file, as shown in Example C-2.

Example: C-2 Sample /etc/ntp.conf file for server with reference clock

```
server 1.2.3.4 prefer #primary server
server 127.127.1.0 #secondary server
driftfile /etc/ntp.drift
tracefile /etc/ntp.trace
```

In this example, 1.2.3.4 is the IP address of the reference clock system.

3. Start the xntp subsystem using the **chrctcp** command, as shown in Example C-3.

Example: C-3 Starting xntpd on the server

```
# /usr/sbin/chrctcp -S -a xntpd
0513-059 The xntpd Subsystem has been started. Subsystem PID is 307392.
#
```

4. Check the status of the client to make sure that it is in contact with the server.

Note: This process can take up to 10 minutes.

Example C-4 shows the output before xntp has contacted the server.

Example: C-4 xntpd status before synchronization

```
# lssrc -ls xntpd
Program name:  /usr/sbin/xntpd
Version:      3
Leap indicator: 11 (Leap indicator is insane.)
Sys peer:     no peer, system is insane
Sys stratum:  16
Sys precision: -17
Debug/Tracing: DISABLED
Root distance: 0.000000
```

```

Root dispersion: 0.000000
Reference ID:   no refid, system is insane
Reference time: no reftime, system is insane
Broadcast delay: 0.003906 (sec)
Auth delay:      0.000122 (sec)
System flags:    bclient pll monitor filegen
System uptime:   10 (sec)
Clock stability: 0.000000 (sec)
Clock frequency: 0.000000 (sec)
Peer: madrid.itsc.austin.ibm.com
      flags: (configured)
      stratum: 2, version: 3
      our mode: client, his mode: server
Peer: rchntp.rchland.ibm.com
      flags: (configured)(preferred)
      stratum: 1, version: 3
      our mode: client, his mode: server
Subsystem      Group      PID      Status
xntpd          tcpip      376860   active

```

Example C-5 shows the output after xntpd has successfully contacted the server.

Example: C-5 xntpd status after synchronization

```

# lssrc -ls xntpd
Program name:   /usr/sbin/xntpd
Version:        3
Leap indicator: 00 (No leap second today.)
Sys peer:      rchntp.rchland.ibm.com
Sys stratum:    2
Sys precision:  -17
Debug/Tracing:  DISABLED
Root distance:  0.045898
Root dispersion: 0.879242
Reference ID:   9.10.225.159
Reference time: c6ab8523.6be8c000 Mon, Aug 15 2005 16:18:27.421
Broadcast delay: 0.003906 (sec)
Auth delay:      0.000122 (sec)
System flags:    bclient pll monitor filegen
System uptime:   279 (sec)
Clock stability: 0.000000 (sec)
Clock frequency: 0.000000 (sec)
Peer: madrid.itsc.austin.ibm.com
      flags: (configured)(sys peer)
      stratum: 2, version: 3
      our mode: client, his mode: server
Peer: rchntp.rchland.ibm.com
      flags: (configured)(sys peer)(preferred)

```


stratum: 1, version: 3			
our mode: client, his mode: server			
Subsystem	Group	PID	Status
xntpd	tcPIP	376860	active

Configuring the NTP server without a reference clock

Figure C-2 illustrates configuring the NTP server with an external reference clock.

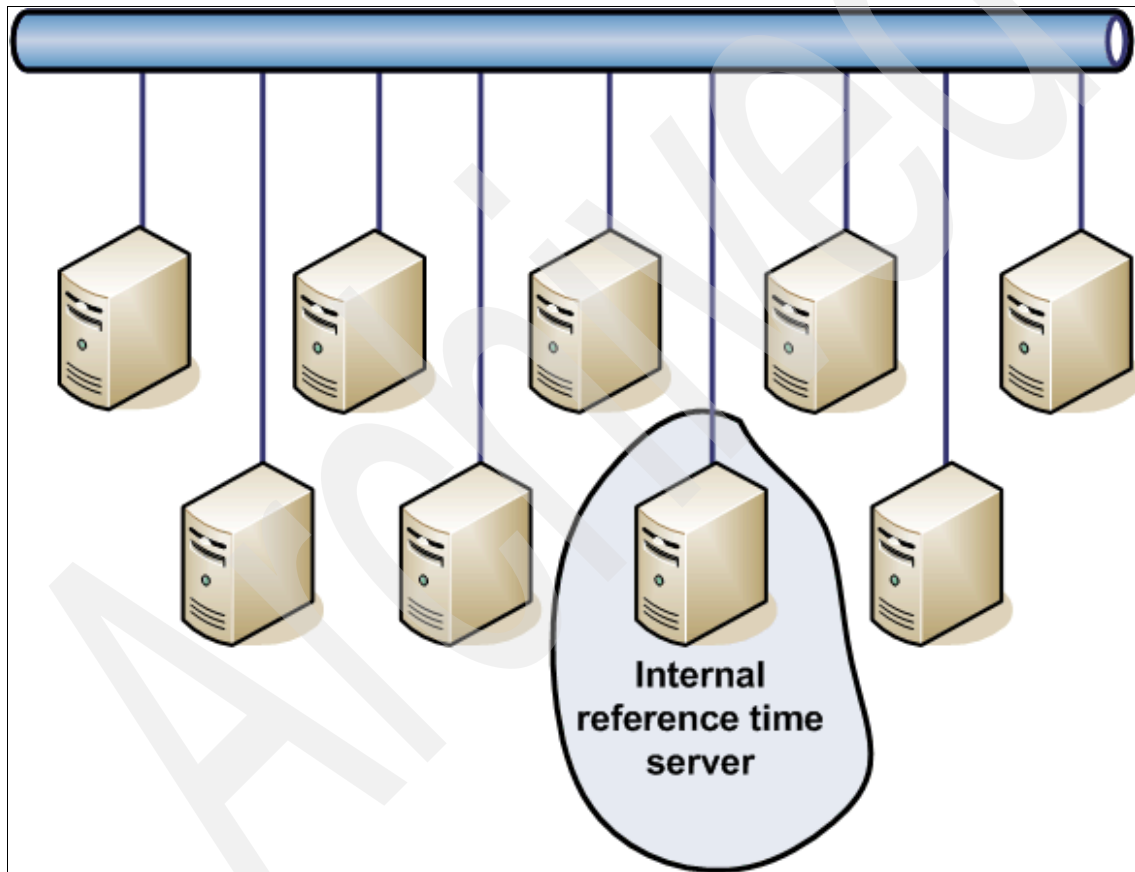


Figure C-2 Configuring NTP without an external reference clock

To set-up an NTP server without an external reference clock, you need to decide which one of your servers will act as the time source for all your systems in your

environment. After making this decision, proceed with the following configuration steps:

1. Make sure that the xntp subsystem is not running, as shown in Example C-6.

Example: C-6 Stopping the xntp subsystem on the server

```
# lssrc -s xntpd
Subsystem      Group      PID      Status
xntpd          tcpip      479410    active
#
# stopsrc -s xntpd
0513-044 The /usr/sbin/xntpd Subsystem was requested to stop.
```

If the subsystem was not running, the first command in Example C-6 would show the status of the daemon as `inoperative`.

2. Edit the `/etc/ntp.conf` file, as shown in Example C-7.

Example: C-7 Sample /etc/ntp.conf file for server without reference clock

```
server 127.127.1.0 prefer #local server
driftfile /etc/ntp.drift
tracefile /etc/ntp.trace
```

3. Start the xntp subsystem using `smit` or the `chrctcp` command, as shown in Example C-8.

Example: C-8 Starting xntpd on the server

```
# /usr/sbin/chrctcp -S -a xntpd
0513-059 The xntpd Subsystem has been started. Subsystem PID is 307392.
#
```

4. Check the status of the client to make sure that it is in contact with the server.

Note: This process can take up to 10 minutes.

Example C-9 shows the output before xntp has contacted the server.

Example: C-9 xntpd status before synchronization

```
# lssrc -ls xntpd
Program name:  /usr/sbin/xntpd
Version:       3
Leap indicator: 11 (Leap indicator is insane.)
Sys peer:     no peer, system is insane
Sys stratum:   16
Sys precision: -17
```

```

Debug/Tracing:  DISABLED
Root distance:  0.000000
Root dispersion: 0.000000
Reference ID:  no refid, system is insane
Reference time: no reftime, system is insane
Broadcast delay: 0.003906 (sec)
Auth delay:      0.000122 (sec)
System flags:    bclient pll monitor filegen
System uptime:   10 (sec)
Clock stability: 0.000000 (sec)
Clock frequency: 0.000000 (sec)
Peer: madrid.itsc.austin.ibm.com
      flags: (configured)
      stratum: 2, version: 3
      our mode: client, his mode: server
Peer: rchntp.rchland.ibm.com
      flags: (configured)(preferred)
      stratum: 1, version: 3
      our mode: client, his mode: server

```

Subsystem	Group	PID	Status
xntpd	tcpip	376860	active

Example C-10 shows the output after xntpd has successfully contacted the server.

Example: C-10 xntpd status after synchronization

```

# lssrc -ls xntpd
Program name:  /usr/sbin/xntpd
Version:      3
Leap indicator: 00 (No leap second today.)
Sys peer:      rchntp.rchland.ibm.com
Sys stratum:   2
Sys precision: -17
Debug/Tracing: DISABLED
Root distance: 0.045898
Root dispersion: 0.879242
Reference ID:  9.10.225.159
Reference time: c6ab8523.6be8c000 Mon, Aug 15 2005 16:18:27.421
Broadcast delay: 0.003906 (sec)
Auth delay:      0.000122 (sec)
System flags:    bclient pll monitor filegen
System uptime:   279 (sec)
Clock stability: 0.000000 (sec)
Clock frequency: 0.000000 (sec)
Peer: madrid.itsc.austin.ibm.com
      flags: (configured)(sys peer)
      stratum: 2, version: 3
      our mode: client, his mode: server

```

```

Peer: rchntp.rchland.ibm.com
  flags: (configured)(sys peer)(preferred)
  stratum: 1, version: 3
  our mode: client, his mode: server
Subsystem      Group      PID      Status
xntpd          tcpip      376860    active

```

Configuring NTP clients

Perform the following steps to configure the NTP clients:

1. Make sure that the xntpd subsystem is not running, as shown in Example C-11.

Example: C-11 Stopping the xntpd subsystem

```

# lssrc -s xntpd
Subsystem      Group      PID      Status
xntpd          tcpip      479410    active
#
# stopsrc -s xntpd
0513-044 The /usr/sbin/xntpd Subsystem was requested to stop.
#

```

If the subsystem was not running, the first command in Example C-11 would show the status of the daemon as inoperative.

2. Edit the /etc/ntp.conf file, as shown in Example C-12.

Example: C-12 Sample /etc/ntp.conf file

```

server 11.22.33.44 prefer #local server
driftfile /etc/ntp.drift
tracefile /etc/ntp.trace

```

In this example, 11.22.33.44 is the IP address of the local server that has been set up.

3. Use the **ntpdate** command to synchronize the client clock with the server, as shown in Example C-13.

Example: C-13 Setting the system time using ntpdate

```

# ntpdate 11.22.33.44
15 Aug 16:12:35 ntpdate[381076]: adjust time server 11.22.33.44offset -0.001624

```

4. Start the xntp subsystem using the **chrctcp** command, as shown in Example C-14.

Example: C-14 Starting xntpd

```
# /usr/sbin/chrctcp -S -a xntpd
0513-059 The xntpd Subsystem has been started. Subsystem PID is 307392.
#
```

5. Check the status of the client to make sure that it is in contact with the server.

Note: This process can take up to 10 minutes.

Example C-15 shows the output before xntp has contacted the server.

Example: C-15 xntp status before synchronization

```
# lssrc -ls xntpd
Program name: /usr/sbin/xntpd
Version: 3
Leap indicator: 11 (Leap indicator is insane.)
Sys peer: no peer, system is insane
Sys stratum: 16
Sys precision: -17
Debug/Tracing: DISABLED
Root distance: 0.000000
Root dispersion: 0.000000
Reference ID: no refid, system is insane
Reference time: no reftime, system is insane
Broadcast delay: 0.003906 (sec)
Auth delay: 0.000122 (sec)
System flags: bclient pll monitor filegen
System uptime: 10 (sec)
Clock stability: 0.000000 (sec)
Clock frequency: 0.000000 (sec)
Peer: madrid.itsc.austin.ibm.com
      flags: (configured)
      stratum: 2, version: 3
      our mode: client, his mode: server
Peer: rchntp.rchland.ibm.com
      flags: (configured)(preferred)
      stratum: 1, version: 3
      our mode: client, his mode: server
Subsystem      Group      PID      Status
xntpd          tcpip      376860    active
```

Example C-16 shows the output after xntp has successfully contacted the server.

Example: C-16 xntp status after synchronization

```
# lssrc -ls xntpd
Program name: /usr/sbin/xntpd
Version: 3
Leap indicator: 00 (No leap second today.)
Sys peer: rchntp.rchland.ibm.com
Sys stratum: 2
Sys precision: -17
Debug/Tracing: DISABLED
Root distance: 0.045898
Root dispersion: 0.879242
Reference ID: 9.10.225.159
Reference time: c6ab8523.6be8c000 Mon, Aug 15 2005 16:18:27.421
Broadcast delay: 0.003906 (sec)
Auth delay: 0.000122 (sec)
System flags: bclient pll monitor filegen
System uptime: 279 (sec)
Clock stability: 0.000000 (sec)
Clock frequency: 0.000000 (sec)
Peer: madrid.itsc.austin.ibm.com
    flags: (configured)(sys peer)
    stratum: 2, version: 3
    our mode: client, his mode: server
Peer: rchntp.rchland.ibm.com
    flags: (configured)(sys peer)(preferred)
    stratum: 1, version: 3
    our mode: client, his mode: server
Subsystem      Group      PID      Status
xntpd          tcpip      376860    active
```

AIX 5L V5.3 NFS quick reference

The purpose of this appendix is to provide you with a quick reference sheet for NFSv4 on AIX 5L V5.3 topics up to and including those introduced in RML03.

We discuss the following topics in this appendix:

- ▶ NFS configuration files
- ▶ NFS daemons
- ▶ NFS commands
- ▶ Export options
- ▶ **mount** command options
- ▶ **nfsd** command options and examples
- ▶ **nfs4cl** command options and examples

NFS configuration files

The following list describes the NFS configuration files:

<code>/etc/rc.nfs</code>	Starts NFS on boot.
<code>/etc/filesystems</code>	Contains file systems to be mounted.
<code>/etc/exports</code>	Contains NFS export definitions.
<code>/etc/xtab</code>	Contains names of file systems currently exported.
<code>/etc/rmtab</code>	Contains names of machines and the file systems they have mounted.
<code>/etc/sm</code>	Directory used by <code>rpc.statd</code> .
<code>/etc/sm.bak</code>	Directory used by <code>rpc.statd</code> .
<code>/etc/state</code>	Directory used by <code>rpc.statd</code> .
<code>/etc/nfs/local_domain</code>	Contains the NFS domain information.
<code>/etc/nfs/hostkey</code>	Specifies the Kerberos host principal and the location of the keytab file.
<code>/etc/nfs/princmap</code>	Maps host names to Kerberos principals when the principal is not the fully qualified domain name of the server.
<code>/etc/nfs/realmd.map</code>	Used by the NFS registry daemon to map incoming Kerberos principals.

NFS daemons

The following list describes the NFS daemons:

<code>/usr/sbin/rpc.lockd</code>	Processes lock requests through the RPC package.
<code>/usr/sbin/rpc.statd</code>	Provides crash-and-recovery functions for the NFS locking services.
<code>/usr/sbin/biod</code>	Sends the client's read and write requests to the server; <i>only runs on the client</i> .
<code>/usr/sbin/rpc.mountd</code>	Answers the requests from clients for file system mounts; <i>only runs on the server</i> .
<code>/usr/sbin/nfsd</code>	Starts the daemons that handle a client's request for file system operations; <i>only runs on the server</i> .

/usr/sbin/portmap	Maps RPC program numbers to Internet port numbers.
/usr/sbin/pcnfsd	Handles service requests from PC-NFS clients.
/usr/sbin/gssd	New daemon for NFSv4; services kernel requests for GSS operations.
/usr/sbin/nfsrgyd	New daemon for NFSv4; provides a name translation service for NFS servers and clients.

NFS commands

The following list describes the NFS commands:

/usr/sbin/mount	Shows what file systems are mounted on the machine on which the command is run, including name of the server and mount options.
/usr/sbin/showmount -e [host]	Shows contents of /etc/xtab file on the [host].
/usr/sbin/showmount -a [host]	Shows the contents of the /etc/rmtab file on the [host].
/usr/sbin/exportfs -va	Exports all file systems defined in /etc/exports and prints the name of each directory as it is exported.
/usr/sbin/exportfs -ua	Unexports all exported directories and prints the name of each directory as it is unexported.
/usr/sbin/mknfs	Configures a system to run NFS and starts NFS daemons.
/usr/sbin/nfso	Configures and lists NFS network options.
/usr/sbin/automount	Mounts an NFS automatically.
/usr/sbin/chnfsexp	Changes the attributes of an NFS exported directory.
/usr/sbin/chnfsmnt	Changes the attributes of an NFS mounted directory.
/usr/sbin/lsnfsexp	Displays the characteristics of directories that are exported with NFS.
/usr/sbin/lsnfsmnt	Displays the characteristics of mounted NFS.

<code>/usr/sbin/mknfsexp</code>	Exports a directory.
<code>/usr/sbin/mknfsmnt</code>	Mounts a directory using NFS.
<code>/usr/sbin/rm nfs</code>	Changes the configuration to stop NFS daemons.
<code>/usr/sbin/rm nfsexp</code>	Removes NFS exported directories from a server's list of exports.
<code>/usr/sbin/chnfsdom</code>	Changes the local NFS domain.
<code>/usr/sbin/nfs4cl</code>	Displays or modifies current NFSv4 statistics and properties.
<code>/usr/sbin/nfshostkey</code>	Configures the host key for an NFS server.
<code>/usr/sbin/chnfsim</code>	Changes the NFS foreign identity mappings.
<code>/usr/sbin/chnfs</code>	Changes the configuration of the system to invoke a specified number of nfsd daemons or to change NFS global configuration values.
<code>/usr/sbin/chnfssec</code>	Changes the default security flavor used by the NFS client.

Export options

The following list describes the **exportfs** options:

<code>-rw</code>	All clients have read/write permission (default).
<code>-ro</code>	All clients have read-only permission.
<code>-rw=Client[:Client]</code>	Exports the directory with read/write permission to the specified clients. Exports the directory read-only to clients not in the list.
<code>-access=Client[:Client]</code>	Gives mount access to each client listed. A client can either be a host name or a net group name.
<code>-root=Client[:Client]</code>	Allows root access from the specified clients.
<code>-vers=version_number[:version_number]</code>	Specifies which versions of NFS are allowed to access the exported directory. Valid versions are 2, 3, and 4.
<code>-exname=external_name</code>	Exports the directory by the specified external name. The <code>external_name</code> must begin with the <code>nfsroot</code> name.

-sec=flavor[:flavor]	Specifies a list of security methods that might be used to access files under the exported directory. Allowable flavor values are sys, dh, none, krb5, krb5i, and krb5p.
-nfsroot	Sets the nfsroot to a specific directory, for example, /exports -nfsroot . (Note that the chnfs utility is the recommended method for setting the nfsroot value.)
-refer=rootpath@host[+host][:rootpath@host[+host]]	A namespace referral will be created at the specified path. The referral directs clients to the specified alternate locations where they can continue operations.
-replicas=rootpath@host[+host][:rootpath@host[+host]]	Replica location information will be associated with the export path. The replica information can be used by NFS version 4 clients to redirect operations to the specified alternate locations if the current server becomes unavailable. The administrator should ensure that appropriate data is available at the replica servers.

mount command options

The following list describes the **-o** options:

ro	Specifies that the mounted file is read-only.
rw	Specifies that the mounted file is read/write accessible (default).
fg	Attempts mount in foreground if first attempt is unsuccessful (default).
bg	Attempts mount in background if first attempt is unsuccessful.
hard	Retries a request until server responds (default).
soft	Returns an error if the server does not respond.
intr	Allows keyboard interrupts on hard mounts.
nointr	Specifies no keyboard interrupts allowed on hard mounts.

acl	Requests using the access control list RPC program for this NFS mount.
sec=[flavor1:...:flavorn]	Specifies a list of security methods that can be used to access files under the mount point. Allowable security flavors are sys, dh, krb5, krb5i, and krb5p.
vers=NFS_version	Specifies the NFS version. Options are 2, 3, and 4. vers=4 is only applicable to AIX 5L V5.3.
cio	Specifies the file system to be mounted for concurrent readers and writers. I/O on files in this file system will behave as though they had been opened with O_CIO specified in the open() system call.
dio	Specifies that I/O on the file system will behave as though all the files had been opened with O_DIRECT specified in the open() system call.

nfso command options and examples

Use the **nfso** command to configure Network File System (NFS) tuning parameters. The **nfso** command sets or displays current or next boot values for NFS tuning parameters. This command can also make permanent changes or defer changes until the next reboot. Whether the command sets or displays a parameter is determined by the accompanying flag. The **-o** flag performs both actions. It can either display the value of a parameter or set a new value for a parameter.

Attention: Use extreme care before changing values using the **nsfo** command. An incorrect change can render the system unusable.

NFSv4 introduces the following new tunable parameters:

utf8	This option allows NFSv4 to perform UTF-8 checking. A value of 1 turns on UTF-8 checking of file names. A value of 0 turns it off.
utf8_validation	Enables checking of file names for the NFSv4 client and server to ensure that they conform to the UTF-8 specification.

nfs_v4_pdt

A value of 1 turns on UTF-8 checking of file names. A value of 0 turns it off.

Sets the number of tables for memory pools used by the biods for NFSv4 mounts.

Use the **vmstat -v** command to look for non-zero values in the client file system I/Os blocked with no fsbuf field.

Increase the number until the blocked I/O count is no longer incremented during workload. The number might need to be increased in conjunction with the **nfs_v4_vm_bufs** option.

nfs_v4_vm_bufs

Sets the number of initial free memory buffers used for each NFSv4 paging device table (PDT) created after the first table. The very first PDT has a set value of 256, 512, 640, or 1000, depending on system memory. This initial value is also the default value of each newly created PDT.

Use the **vmstat -v** command to look for non-zero values in the client file system I/Os blocked with no fsbuf field.

The **nfs_v4_vm_bufs** option must be set prior to the **nfs_v4_pdt** option.

The following options are available on AIX 5L V5.3 RML03:

nfs_v4_fail_over_timeout

Specifies how long the NFS client will wait (in seconds) before switching to another server when data is replicated and the current associated server is not accessible. If the default value of 0 is set, the client dynamically determines the timeout as twice the RPC call timeout that was established at mount time or with **nfs4cl**. The **nfs_v4_fail_over_timeout** option is client-wide; if set, the **nfs_v4_fail_over_timeout** option overrides the default behavior on all replicated data.

A value of 0 allows the client to internally determine the timeout value. A positive value overrides the default and specifies the replication fail-over timeout in seconds for all data accessed by the client.

server_delegation	<p>Enables or disables NFS version 4 server delegation support.</p> <p>A value of 1 enables server delegation support. A value of 0 disables server delegation support. Server delegation can also be controlled by using the <code>/etc/exports</code> file and the exportfs command.</p>
client_delegation	<p>Enables or disables NFS version 4 client delegation support.</p> <p>A value of 1 enables client delegation support. A value of 0 disables client delegation support.</p>

The following list provides useful examples about how you can use the **nfso** command:

- ▶ To print, in colon-delimited format, a list of all tunable parameters and their current values, run:


```
nfso -a -c
```
- ▶ To list the current and reboot value, range, unit, type, and dependencies of all tunable parameters managed by the **nfso** command, run:


```
nfso -L
```
- ▶ To list the reboot values for all NFS tuning parameters, run:


```
nfso -r -a
```
- ▶ To set a tunable parameter, for example, `utf8`, to a value of 1, run:


```
nfso -o utf8=1
```
- ▶ To set a tunable parameter, for example, `nfs_v4_pdts`, to its default value of 1 at the next reboot, run:


```
nfso -r -d nfs_v4_pdts
```

The following references are also useful:

- ▶ The “Network File System (NFS) Overview for System Management” section in *AIX 5L Version 5.3 System Management Guide: Communications and Networks*, SC23-4909
- ▶ The “TCP/IP Overview for System Management” section in *AIX 5L Version 5.3 System User's Guide: Communications and Networks*, SC23-4912
- ▶ The “Monitoring and Tuning NFS Use” section in *AIX 5L Version 5.3 Performance Management Guide*, SC23-4876

nfs4cl command options and examples

The **nfs4cl** command is used to display all the file system ID (fsid) information about the client or modify file system options of an fsid.

Note: The **nfs4cl** command updates affect newly accessed files in the file system. An unmount and remount is required to affect all previously accessed files.

The syntax of the **nfs4cl** command is as follows:

```
/usr/sbin/nfs4cl [subcommand] [path] [argument]
```

The following list describes the subcommands and arguments of the **nfs4cl** command:

resetfsoptions subcommand

This subcommand resets all the options for the FSID back to the default options. Note that the **cio** and **dio** options can be reset with the **resetfsoptions** subcommand, but the **cio** and **dio** behavior is not turned off until the NFS file system is unmounted and then remounted.

setfsoptions subcommand

This subcommand takes a path and an argument. The path specifies the target fsid structure, and the argument is the file system options. It sets the internal fsid to use the options specified by the argument. Here is the list of possible arguments:

rw	Specifies that the files or directories that bind to this path (fsid) are readable and writable.
ro	Specifies that the files or directories that bind to this path (fsid) are read-only.
acdirmax	Specifies the upper limit for the directory attribute cache timeout value.
acdirmin	Specifies the lower limit for the directory attribute cache timeout value.
acregmax	Specifies the upper limit for the file attribute cache timeout value.
acregmin	Specifies the lower limit for the file attribute cache timeout value.

cio	Specifies the file system to be mounted for concurrent readers and writers. I/O on files in this file system behave as though the file was opened with O_CIO specified in the open() system call.
dio	Specifies that I/O on the file system behaves as though all of the files were opened with O_DIRECT specified in the open() system call.
hard	Specifies that this fsid will use hard mount semantics.
intr	Specifies that the fsid operations are interruptible.
maxpout=value	Specifies the pageout level for files on this file system at which threads should be slept. If maxpout is specified, minpout must also be specified. This value must be non-negative and greater than minpout. The default is the kernel maxpout level.
minpout=value	Specifies the pageout level for files on this file system at which threads should be readied. If minpout is specified, maxpout must also be specified. This value must be non-negative. The default is the kernel minpout level.
noac	Does not use attribute cache.
nocto	Specifies no close-to-open consistency.
nointr	Specifies that the fsid is non-interruptible.
prefer	Administratively sets the preferred server to use when data exists at multiple server locations.
rbr	Uses the release-behind-when-reading capability. When sequential reading of a file in this file system is detected, the real memory pages used by the file will be released after the pages are copied to internal buffers.
rsize	Specifies the read size for the RPC calls to the server.
retrans	Specifies the number of RPC retransmits to attempt with soft semantics.
soft	Specifies the fsid operation that will use soft mount semantics.
timeo	Specifies the time out value for the RPC calls to the server.
wsiz	Specifies the write size for the RPC calls to the server.
nodircahe	Does not use directory cache.

showfs subcommand

This subcommand displays file system-specific information about the server that is currently accessed by the client. The information includes server address, remote path, fsid, and local path. If path is provided, additional information, such as fs_locations and fsid options, are displayed.

showstats subcommand

This subcommand shows information similar to what the **df** command prints out for each fsid that exists on the client. The information includes fields such as File system, 512- blocks, Free, %Used, lused, %lused, and Mounted on.

Examples of nfs4cl usage

The following list provides examples of the **nfs4cl** command:

- To display all the fsid structure on the client, run:

```
nfs4cl showfs
```

Example D-1 shows the sample output from the **nfs4cl showfs** command.

Example: D-1 Sample output from the nfs4cl showfs command

```
# nfs4cl showfs
```

Server	Remote Path	fsid	Local Path
-----	-----	-----	-----
sabine	/gpfs1	2:150	/mnt/gpfs1

- To set the file system options of /mnt/usr/sbin to include only retrans=3, run:

```
nfs4cl setfsoptions /mnt/gpfs1 retrans=3
```

See Example D-2.

Example: D-2 nfs4cl command used to change the retrans option on a client

```
# nfs4cl showfs /mnt/gpfs1
```

Server	Remote Path	fsid	Local Path
-----	-----	-----	-----
sabine	/gpfs1	2:150	/mnt/gpfs1

```
Current Server: sabine:/mnt/gpfs1
Replica Server: frio:/mnt/gpfs1
Replica Server: angelina:/mnt/gpfs1
```

options :

```
rw,intr,rsz=32768,wsz=32768,timeo=100,retrans=5,maxgroups=0,acregmin=3,acregmax=60,acdirmin=30,acdirmax=60,minpout=1250,maxpout=2500,sec=sys:krb5:krb5i:krb5p
```

```
#
# nfs4cl setfsoptions /mnt/gpfs1 retrans=3
#
# nfs4cl showfs /mnt/gpfs1
```

Server	Remote Path	fsid	Local Path
-----	-----	-----	-----
sabine	/gpfs1	2:150	/mnt/gpfs1
Current Server: sabine:/mnt/gpfs1			
Replica Server: frio:/mnt/gpfs1			
Replica Server: angelina:/mnt/gpfs1			
options :			
rw,intr,rsiz=32768,wsiz=32768,timeo=100, retrans=3 ,maxgroups=0,acregmin=3,acre			
gmax=60,acdirmin=30,acdirmax=60,minpout=1250,maxpout=2500,sec=sys:krb5:krb5i:kr			
b5p			

- To reset the file system options for /mnt/gpfs1, run:

```
nfs4cl resetfsoptions /mnt/gpfs1
```

See Example D-3.

Example: D-3 nfs4cl command used to reset file system options

```
# nfs4cl showfs /mnt/gpfs1
```

Server	Remote Path	fsid	Local Path
-----	-----	-----	-----
sabine	/gpfs1	2:150	/mnt/gpfs1
Current Server: sabine:/mnt/gpfs1			
Replica Server: frio:/mnt/gpfs1			
Replica Server: angelina:/mnt/gpfs1			
options :			
rw,intr,rsiz=32768,wsiz=32768,timeo=100, retrans=3 ,maxgroups=0,acregmin=3,acre			
gmax=60,acdirmin=30,acdirmax=60,minpout=1250,maxpout=2500,sec=sys:krb5:krb5i:kr			
b5p			

```
#
# nfs4cl resetfsoptions /mnt/gpfs1
#
# nfs4cl showfs /mnt/gpfs1
```

Server	Remote Path	fsid	Local Path
-----	-----	-----	-----
sabine	/gpfs1	2:150	/mnt/gpfs1
Current Server: sabine:/mnt/gpfs1			
Replica Server: frio:/mnt/gpfs1			
Replica Server: angelina:/mnt/gpfs1			
options :			
rw,nointr,rsiz=32768,wsiz=32768,timeo=100, retrans=5 ,maxgroups=0,acregmin=3,ac			

regmax=60,acdirmin=30,acdirmax=60,minpout=2500,maxpout=1250,sec=sys:krb5:krb5i:krb5p

- ▶ To show **df** command output for /mnt/gpfs1, run:

nfs4cl showstat /mnt/gpfs1

See Example D-4.

Example: D-4 nfs4cl command used to display df command output

nfs4cl showstat /mnt/gpfs1

Filesystem	512-blocks	Free	%Used	Iused	%Iused	Mounted on
sabine:/gpfs1	2293323776	2292447232	1%	64	1%	/mnt/gpfs1

#

- ▶ To change the preferred NFSv4 server on a client, run:

nfs4cl setfsoptions /mnt/gpfs1 prefer=angelina:/gpfs1

See Example D-5.

Example: D-5 nfs4cl command used to change the preferred NFSv4 server on a client

nfs4cl setfsoptions /mnt/gpfs1 prefer=angelina

#

nfs4cl showfs /mnt/gpfs1

Server	Remote Path	fsid	Local Path
-----	-----	-----	-----
sabine	/gpfs1	2:150	/mnt/gpfs1

Current Server: sabine:/mnt/gpfs1

Replica Server: frio:/mnt/gpfs1

Replica Server: angelina:/mnt/gpfs1

options :

rw,nointr,rsz=32768,wsz=32768,timeo=100,retrans=5,maxgroups=0,acregmin=3,acregmax=60,acdirmin=30,acdirmax=60,minpout=2500,maxpout=1250,sec=sys:krb5:krb5i:krb5p,prefer=angelina.itsc.austin.ibm.com

Scripts and configuration files

This appendix provides listings of various scripts and configuration files used during the test scenarios developed for this book.

This appendix contains the following scripts and configuration files:

- ▶ LDAP schema for the KDC realm
- ▶ Script to add users to the KDC
- ▶ Script to migrate DFS to AIXC ACLs
- ▶ Script to migrate DFS to NFSv4 ACLs
- ▶ Script migusr to migrate users from AFS to Kerberos/LDAP
- ▶ Script migrate_afs_groups.pl to migrate groups: AFS to Kerberos/LDAP
- ▶ Script to migrate AFS ACLs to NFSv4
- ▶ Script to generate mkgroup commands using DCE group data
- ▶ Script to generate mkuser commands using DCE user account data
- ▶ Script to copy an ACL (with recursive option)

Sample LDAP LDIF file for the KDC realm

Example E-1 provides a sample LDAP LDIF file for the KDC realm.

Example: E-1 LDAP schema for the KDC realm

```
##
## Copyright 2005 IBM Corporation. All rights reserved. This script
## is provided solely as an example for migration planning purposes;
## no warranty, expressed or implied, is provided for this utility.
##
# version: 1
dn: o=IBM, c=US
objectclass: top
objectclass: organization
o: IBM

dn: krbrealmName-V2=NFSV4REALM.IBM.COM, o=IBM, c=US
objectclass: KrbRealm-V2
objectclass: KrbRealmExt
krbrealmName-V2: NFSV4REALM.IBM.COM
krbprincSubtree: krbrealmName-V2=NFSV4REALM.IBM.COM, o=IBM, c=US
krbDeleteType: 3

dn: cn=principal, krbrealmName-V2=NFSV4REALM.IBM.COM, o=IBM, c=US
objectclass: container
cn: principal

dn: cn=policy, krbrealmName-V2=NFSV4REALM.IBM.COM, o=IBM, c=US
objectclass: container
cn: policy
```

Script to add users to the KDC

Example E-2 provides a script to add users to the KDC.

Example: E-2 Script to add users to the KDC

```
#!/bin/ksh
##
## Copyright 2005 IBM Corporation. All rights reserved. This script
## is provided solely as an example for migration planning purposes;
## no warranty, expressed or implied, is provided for this utility.
##
# Script to migrate user information. Script name migusr
```

```

# The script does the following:
#
# i. adds the user
# ii. displays user information in LDAP
# iii. sets the user's temporary password
# iv. expires the user's password, forcing a change at first log on
#
export AUTH=KRB5LDAP
export PASSWD=tempONETemp

# if the admusr user does not exist, create it.
lsuser -R $AUTH admusr || /usr/bin/mkuser -R $AUTH -a account_locked=true
admin=true admusr

# extract the username, UID and gecos information from the input file
cat /mnt/user_data.out | awk -F":" '

# add user routine with UID & gecos info
adduser=sprintf("/usr/bin/mkuser -R $AUTH -a id=%s gecos=\"%s\" %s", $2, $3,
$1)

# display user information from LDAP routine
lsuser=sprintf("/usr/sbin/lsuser -R $AUTH %s", $1)

# set the the temporary password routine
passwd=sprintf("/usr/krb5/sbin/kadmin.local -q \"change_password -pw $PASSWD
%s\"", $1)

# expire the temporary password routine
chuser=sprintf("/usr/bin/chuser -R $AUTH krb5_attributes=+needchange %s", $1)

# run the routines
system(adduser)
system(lsuser)
system(passwd)
system(chuser)
# single quote that follows ends the awk statement...
'

```

DFS to AIXC ACL migration example

Example E-3 provides a script to migrate DFS to AIXC ACLs.

Example: E-3 Script to migrate DFS to AIXC ACLs

```
#!/usr/bin/perl

## migrate_dfs_acls_to_aixc.pl -- Example ACL migration script
## showing the steps required to copy existing DFS ACL information
## into a format usable by AIXC 'aclput' commands.
##
## Copyright 2005 IBM Corporation. All rights reserved. This script
## is provided solely as an example for migration planning purposes;
## no warranty, expressed or implied, is provided for this utility.
##
## Theory of Operation.
## The script uses standard methods to recursively read a user's
## entire directory tree, executing "dcecp -c acl show" commands on
## each file and sub-directory and using the results to generate AIX-
## style ACL input files in the directory $acl_directory. At the
## same time it generates a shell script, <cwd>/acl_<user_id>.sh,
## containing a series of aclput -i commands referencing the files
## found in $acl_directory and the fully qualified file path, e.g.
## "aclput -i <cwd>/$acl_directory/filename.txt.acl <cwd>/filename.txt"
##
## Assumptions.
##
## The base cell name (obtained using dcecp -c getcellname) is
## trimmed from the CWD in order to restructure the name space;
## if the current cell is ../../test.itsc.austin.ibm.com, then
## this string, along with the "/fs" DFS root, will be removed
## from all path names. The --prefix switch can be used to alter
## the new root directory to /nfs/ or any other value (the default
## is / if the --prefix switch is no specified).
##
## We assume the directory structure has been copied intact from
## the DFS cell to the new AIX environment; deviations from this
## basic methodology can be handled by manually editing the output
## shell script or altering the base Perl code.
##
## Notes & caveats
##
## 1) It must be run from a user's home directory while that user
## is logged in under their own id. The script makes use of
## the $HOME and other variables when formatting ACLs and
## directory paths.
##
## 2) No handling is provided for situations where duplicate file
```



```

##      names exist in a directory tree (e.g. $HOME/dir1/file.txt
##      and $HOME/dir2/file.txt). In this case, the last file
##      in the tree with the duplicate name will be used to build
##      the ACL.
##

use Cwd; # module for finding the current working directory
use Getopt::Long qw(GetOptions);

my $user = $ENV{"LOGIN"};
my $home = $ENV{"HOME"};
my $parsed_home = "";
my $acl_directory = "";
my $acl_base = "acl_data";
my $output_file = "";
my ($login,$pass,$uid,$gid) = getpwnam($user);
my $group = getgrgid($gid);
my $do_symlinks = ''; ## by default we'll ignore symlinks
my $prefix = ""; ## default
my $cell_name = "/...";

$|=1;    # turn off I/O buffering

sub ScanDirectory {
    my ($workdir, $output_file, $prefix, $cell_name) = @_;

    ## we should ignore both the "acl_base" directory and OldFiles
    return if ($workdir eq $acl_base);
    return if ($workdir eq "OldFiles");

    my($startdir) = &cwd; # keep track of where we began

    chdir($workdir) or die "Unable to enter dir $workdir:!\n";

    opendir(DIR, ".") or die "Unable to open $workdir:!\n";
    my @names = readdir(DIR);
    closedir(DIR);

    foreach my $name (@names){
        next if ($name eq ".");
        next if ($name eq "..");

        if (-d $name){                # is this a directory?
            &ScanDirectory($name, $output_file, $prefix, $cell_name);
            next;
        }
        unless (&CheckFile($name, $output_file, $prefix, $cell_name)){
            print &cwd."/".$name."\n"; # print the bad filename
        }
    }
}

```

```

    }
    chdir($startdir) or die "Unable to change to dir $startdir:!\n";
}

sub CheckFile{
    my($name, $output_file, $prefix, $cell_name) = @_;

    ## use the FQ path when checking the existing file
    my $file_to_check = &cwd."/".$name;
    ## now generate a new path to this subdirectory once it's been
    ## migrated to AIX.
    my $current_dir = &cwd;
    $current_dir =~ s/$cell_name//;
    my $destination_file = $prefix."/".$current_dir."/".$name;
    $destination_file =~ s/\\/\\\\/g; ## remove doubled "/"

    ## If this file is a symbolic link and we we turned off processing
    ## of these files, then we'll just return from this function.

    return if ( (-l $file_to_check) && ($do_symlinks eq '') );
    print "Traversing symlink: $file_to_check\n" if (-l $file_to_check);

    ## attempt to read the directory entry for this file
    print STDERR "Scanning $file_to_check\n";

    ## Try to open the file. If we fail, we'll just drop out of the
    ## loop and try the next one. Sites should decide if this is
    ## what they want to do, or if they prefer other processing to
    ## occur in such a case.
    if (open (ACL_DATA, "dcecp -c acl show $file_to_check 2>&1 |") ) {

        ## First we'll open a file in which to write AIX ACL commands,
        ## then we'll read data a line at a time from the return
        open (ACL_FILE, ">$acl_directory/$name.acl") ||
            die "failed to open output $acl_directory/$name.acl: $!\n";
        print ACL_FILE "attributes:\nbased permissions\n"; ## file preamble

        ## Next we write our aclput command to our master script.
        print OUTFILE "aclput -i $new_acl_dir/$name.acl $destination_file\n";

        ## now we loop through returned data to parse the DCE ACLs into
        ## a form usable by the AIX 'aclput' utility. There are two
        ## possible results: {user|group|other|mask}_obj + permissions, or
        ## {user|group} + user/group name + permissions. The former will
        ## convert into the usual rwx format, while all others need to
        ## be translated to an AIX equivalent (if it exists).

    }

    my $extended_acls = "disabled"; ## false by default
    my @base_perms = @ext_perms = ();

```

```

while (my $data = <ACL_DATA>) {
  if ($data =~ /_obj/) {
    my ($user_id, $perm_mask) = split " ", $data;
    my ($bit1, $bit2, $bit3, $bit4, $bit5, $bit6) = split "", $perm_mask;

    push @base_perms, "    owner($user): $bit1$bit2$bit3\n"
      if ($data =~ /user_obj/);

    push @base_perms, "    group($group): $bit1$bit2$bit3\n"
      if ($data =~ /group_obj/);

    push @base_perms, "    others: $bit1$bit2$bit3\n"
      if ($data =~ /other_obj/);
  } else {

    ## the first time we encounter an extended acl on any given
    ## file, we'll need to change $extended_acls to "enabled" and
    ## write it, along with the "extended permissions" line, as
    ## the next line in the file. Otherwise it'll remain set as
    ## "disabled."

    $extended_acls = "enabled" if ($extended_acls eq "disabled");

    my ($type, $user_id, $perm_mask) = split " ", $data;
    $type =~ s/{//; ## remove leading "{"
    $type = "g" if ($type eq "group");
    $type = "u" if ($type eq "user");
    my ($bit1, $bit2, $bit3, $bit4, $bit5, $bit6) = split "", $perm_mask;

    push @ext_perms, "    permit  $bit1$bit2$bit3    $type:$user_id\n";
  } ## end of if-else
} ## end of while loop

my $line = "";
## now, write all the data to the output file...

print ACL_FILE $line while ($line = pop(@base_perms));
$line = "";
print ACL_FILE "extended permissions\n    $extended_acls\n";

if ($extended_acls eq "enabled") {
  print ACL_FILE $line while ($line = pop(@ext_perms));
}
close ACL_FILE;
close ACL_DATA || warn "dcecp error: $! $?";
$extended_acls = "disabled"; ## paranoia
}

```

```

return 1;
}

sub print_help
{
    print "Usage: perl migrate_dfs_acls_to_aixc.pl [--help] [--symlinks]\n
[--prefix=<new_namespace_prefix>]\n";
    print "    --symlinks: the script will traverse symlinks (default: off)\n";
    print "    --prefix: specifies the top-level directory prefix in use in\n";
    print "        the AIX namespace. This will replace the existing DFS\n";
    print "        \"/.../existing.cell.name/fs\" prefix (default: /) \n";
    print "    --help: prints this help text\n";
    exit(0);
}

## main logic

GetOptions ( 'symlinks' => \$do_symlinks,
             'help' => \$print_help,
             'prefix=s' => \$prefix);

&print_help() if ($print_help);

## Get the current DCE cell name
if (open (CELL_DATA, "dcecp -c getcellname 2>&1 |") ) {
    my $data = <CELL_DATA>; ## read only 1 line
    $cell_name = $data;
} else {
    print "Error retrieving cell name: $!\n";
    exit(1);
}
close CELL_DATA;

$cell_name =~ s/\n\/fs/; ## strip carriage return and add DFS /fs suffix
$parsed_home = $home; ## need both
$parsed_home =~ s/$cell_name//;

## Only prepend a prefix if it's non-null.
$parsed_home = "$prefix/$parsed_home" if ($prefix ne "");

## New base directory
$parsed_home =~ s/\/\/\/\/g; ## remove any doubled "/"
## location of .sh file in current cell
$output_file = ".acl_$user.sh"; ## start at CWD in existing cell

## Set up "destination" directories
$acl_directory = $home."/". $acl_base;

```

```

$new_acl_dir = $parsed_home."/".$acl_base;

## Emit our expected parameters
print "Starting at base directory: $home.\nDestination directory:
$parsed_home\n";

## location for acl output files in current cell
system("mkdir $acl_directory") if (!-e "$acl_directory");

open (OUTFILE, ">$output_file");
print OUTFILE "#!/bin/sh\n";

&ScanDirectory(".", $output_file, $prefix, $cell_name);

close OUTFILE;
exit(0);

```

DFS to NFSv4 migration example

Example E-4 provides a script to migrate DFS to NFSv4 ACLs.

Example: E-4 Script to migrate DFS to NFSv4 ACLs

```

#!/usr/bin/perl

## migrate_dfs_acls_to_nfsv4.pl -- Example ACL migration script designed
## to show the steps required to copy existing DFS ACL information
## into a format usable by NFSv4 'aclput' commands.
##
## Copyright 2005 IBM Corporation. All rights reserved. This script
## is provided solely as an example for migration planning purposes;
## no warranty, expressed or implied, is provided for this utility.
##
## Theory of Operation.
## The script uses standard methods to recursively read a user's
## directory tree, executing "dcecp -c acl show" commands on each
## file and sub-directory and using the results to generate NFSv4-
## style ACL input files in the directory $acl_directory. At the
## same time it generates a shell script, <cwd>/acl_<user_id>.sh,
## containing a series of aclput -i commands referencing the files
## found in $acl_directory and the fully qualified file path, e.g.
## "aclput -i <cwd>/$acl_directory/filename.txt.acl <cwd>/filename.txt"
##
## Assumptions.
##
## The base cell name (obtained using dcecp -c getcellname) is
## trimmed from the CWD in order to restructure the namespace;

```

```

## if the current cell is ../../test.itsc.austin.ibm.com, then
## this string, along with the "/fs" DFS root, will be removed
## from all path names. The --prefix switch can be used to alter
## the new root directory to /nfs/ or any other value (the default
## is / if the --prefix switch is no specified).
##
## We assume the directory structure has been copied intact from
## the DFS cell to the new NFSv4 environment; deviations from this
## basic methodology can be handled by manually editing the output
## shell script or altering the base Perl code.
##
## Notes & caveats
##
## 1) It must be run from a user's home directory while that user
##    is logged in under their own id. The script makes use of
##    the $HOME and other variables when formatting ACLs and
##    directory paths.
##
## 2) No handling is provided for situations where duplicate file
##    names exist in a directory tree (e.g. $HOME/dir1/file.txt
##    and $HOME/dir2/file.txt). In this case, the last file
##    in the tree with the duplicate name will be used to build
##    the ACL.
##

use Cwd; # module for finding the current working directory
use Getopt::Long qw(GetOptions);

my $user = $ENV{"LOGIN"};
my $home = $ENV{"HOME"};
my $main_group = $ENV{"GROUP"};
my $parsed_home = "";
my $acl_directory = "";
my $acl_base = "acl_data";
my $output_file = "";
my ($login,$pass,$uid,$gid) = getpwnam($user);
my $group = getgrgid($gid);
my $do_symlinks = ''; ## by default we'll ignore symlinks
my $prefix = ""; ## default
my $cell_name = "/...";

$|=1;    # turn off I/O buffering

sub ScanDirectory {
    my ($workdir, $output_file, $prefix, $cell_name) = @_;

    ## we should ignore both the "acl_base" directory and OldFiles
    return if ($workdir eq $acl_base);
    return if ($workdir eq "OldFiles");

```

```

my($startdir) = &cwd; # keep track of where we began

chdir($workdir) or die "Unable to enter dir $workdir:!\n";

opendir(DIR, ".") or die "Unable to open $workdir:!\n";
my @names = readdir(DIR);
closedir(DIR);

foreach my $name (@names){
    next if ($name eq ".");
    next if ($name eq "..");

    if (-d $name){ # is this a directory?
        &ScanDirectory($name, $output_file, $prefix, $cell_name);
        next;
    }
    unless (&CheckFile($name, $output_file, $prefix, $cell_name)){
        print &cwd."/".$name."\n"; # print the bad filename
    }
}
chdir($startdir) or die "Unable to change to dir $startdir:!\n";
}

sub CheckFile{
    my($name, $output_file, $prefix, $cell_name) = @_;

    ## use the FQ path when checking the existing file
    my $file_to_check = &cwd."/".$name;

    ## now generate a new path to this subdirectory once it's been
    ## migrated to NFSv4.
    my $current_dir = &cwd;
    $current_dir =~ s/$cell_name//;
    my $destination_file = $prefix."/".$current_dir."/".$name;
    $destination_file =~ s/\//\\/g; ## remove doubled "/"

    ## If this file is a symbolic link and we turned off processing
    ## of these files, then we'll just return from this function.
    return if ( (-l $file_to_check) && ($do_symlinks eq '') );
    print "Traversing symlink: $file_to_check\n" if (-l $file_to_check);

    ## attempt to read the directory entry for this file
    print STDERR "Scanning $file_to_check\n";

    ## Try to open the file. If we fail, we'll just drop out of the
    ## loop and try the next one. Sites should decide if this is
    ## what they want to do, or if they prefer other processing to
    ## occur in such a case.

```

```

if (open (ACL_DATA, "dcecp -c acl show $file_to_check 2>&1 |") ) {

    ## First we'll open a file in which to write NFSv4 ACL commands,
    ## then we'll read data a line at a time from the return
    open (ACL_FILE, ">$acl_directory/$name.acl") ||
        die "failed to open output $acl_directory/$name.acl: $!\n";

    print ACL_FILE "*\n* ACL_type   NFS4\n*\n*\n* Owner: $user\n* Group:
$group\n*\n*";
    ## Next we write our aclput command to our master script.
    print OUTFILE "aclput -i $new_acl_dir/$name.acl $destination_file\n";

    ## now we loop through returned data to parse the DCE ACLs into
    ## a form usable by the NFSv4 'aclput' utility. There are two
    ## possible results: {user|group|other|mask}_obj + permissions, or
    ## {user|group} + user/group name + permissions. The former will
    ## be written to the file using a type::acl format, while the rest
    ## need to appear as type:id:acl, where "id" is either user or
    ## group.

    my @base_perms = @ext_perms = ();
    while (my $data = <ACL_DATA>) {
        $data =~ s/[{}]/ /g; ## remove leading & trailing braces
        if ($data =~ /_obj/) {
            my ($user_id, $perm_mask) = split " ", $data;

            ## we need the ability to work on discrete permission bits;
            ## to do this, we'll transform the permissions list into
            ## an array.

            my ($bit1, $bit2, $bit3, $bit4, $bit5, $bit6) = split "", $perm_mask;
            my @perms = ($bit1, $bit2, $bit3, $bit4, $bit5, $bit6);

            ## next step is the actual conversion of a DCE ACL to an
            ## NFSv4 equivalent. In some cases, a single bit may need
            ## to become a 2- or even 3-bit equivalents in NFSv4 (e.g.
            ## a DFS "r" becomes "ra", and we will also give the user
            ## the proper access to named attributes such as RAW).

            ## Important: this example script will not handle all the
            ## possible ACL permutations and only creates "allow" (a)
            ## ACL entries. Further development is left as an exercise
            ## for individual sites requiring a customized solution.

            my $acl_string = "";

            for (my $i=0; $i<=5;$i++) {
                my $temp = "";
                $temp = "RAra" if ($perms[$i] eq "r");

```



```

    $temp = "Wwp" if ($perms[$i] eq "w");
    $temp = "x" if ($perms[$i] eq "x");
    $temp = "Dd" if ($perms[$i] eq "d");
    $temp = "cC" if ($perms[$i] eq "c");
    $acl_string = "$acl_string$temp";
}
## Next, we start populating arrays that will later be used
## to add the data to our ACL input file.

push @base_perms, "s:(OWNER@):    a        $acl_string\n"
    if ($data =~ /user_obj/);

push @base_perms, "s:(GROUP@):    a        $acl_string\n"
    if ($data =~ /group_obj/);

push @base_perms, "s:(EVERYONE@): a        $acl_string\n"
    if ($data =~ /other_obj/);

} else {

    ## We use a different split when we encounter a non-generic
    ## acl on a given file.

    my ($type, $user_id, $perm_mask) = split " ", $data;
    my ($bit1, $bit2, $bit3, $bit4, $bit5, $bit6) = split "", $perm_mask;
    my @perms = ($bit1, $bit2, $bit3, $bit4, $bit5, $bit6);

    $type = "g" if ($type eq "group");
    $type = "u" if ($type eq "user");

    ## Again, we'll transform the ACLs. This work is better handled
    ## in a subroutine, but for this example we'll leave it in-line
    ## for clarity. Here we do not give controlling (RAW) access
    ## since it's not clear this would be desirable.

    my $acl_string = "";

    for (my $i=0; $i<=5;$i++) {
        my $temp = "";
        $temp = "ra" if ($perms[$i] eq "r");
        $temp = "wp" if ($perms[$i] eq "w");
        $temp = "x" if ($perms[$i] eq "x");
        $temp = "d" if ($perms[$i] eq "d");
        $temp = "cC" if ($perms[$i] eq "c");
        $acl_string = "$acl_string$temp";
    }

    push @ext_perms, "$type:$user_id:    a        $acl_string\n";

```

```

    } ## end of if-else
} ## end of while loop

my $line = "";
## now, write all the data to the output file...they're likely to
## be stored in reverse order so we'll loop through the array
## rather than use pop to pull them off in reverse order.

for(my $counter=0 ; $counter < scalar(@ext_perms) ; $counter++) {
    my $line = $ext_perms[$counter];
    print ACL_FILE $line;
}
for(my $counter=0 ; $counter < scalar(@base_perms) ; $counter++) {
    my $line = $base_perms[$counter];
    print ACL_FILE $line;
}
$line = "";

close ACL_FILE;
close ACL_DATA || warn "dcecp error: $! $?";
}

return 1;
}

sub print_help
{
    print "Usage: perl migrate_dfs_acls_to_aixc.pl [--help] [--symlinks]\n";
    print "--prefix=<new_namespace_prefix>\n";
    print "    --symlinks: the script will traverse symlinks (default: off)\n";
    print "    --prefix: specifies the top-level directory prefix in use in\n";
    print "               the NFSv4 namespace. This will replace the existing DFS\n";
    print "               \"/.../existing.cell.name/fs\" prefix (default: /) \n";
    print "    --help: prints this help text\n";
    exit(0);
}

## main logic

GetOptions ( 'symlinks' => \$do_symlinks,
             'help' => \$print_help,
             'prefix=s' => \$prefix);

&print_help() if ($print_help);

```

```

## Get the current DCE cell name
if (open (CELL_DATA, "dcecp -c getcellname 2>&1 |") ) {
    my $data = <CELL_DATA>; ## read only 1 line
    $cell_name = $data;
} else {
    print "Error retrieving cell name: $!\n";
    exit(1);
}
close CELL_DATA;

$cell_name =~ s/\n\\/fs/; ## strip carriage return and add DFS /fs suffix
$parsed_home = $home; ## need both
$parsed_home =~ s/$cell_name//;

## Only prepend a prefix if it's non-null.
$parsed_home = "$prefix/$parsed_home" if ($prefix ne "");

## New base directory
$parsed_home =~ s/\\/\\/\\/g; ## remove any doubled "/"

## location of .sh file in current cell
$output_file = "./acl_$user.sh"; ## start at CWD in existing cell

## Set up "destination" directories
$acl_directory = $home."/".$acl_base;
$new_acl_dir = $parsed_home."/".$acl_base;

## Emit our expected parameters
print "Starting at base directory: $home.\nDestination directory:
$parsed_home\n";

## location for acl output files in current cell
system("mkdir $acl_directory") if (!-e "$acl_directory");

open (OUTFILE, ">$output_file");
print OUTFILE "#!/bin/sh\n";

&ScanDirectory(".", $output_file, $prefix, $cell_name);

close OUTFILE;
exit(0);

```

AFS to Kerberos/LDAP user migration

Example E-5 provides a script to migrate users from AFS to Kerberos/LDAP.

Example: E-5 Script migusr to migrate users from AFS to Kerberos/LDAP

```
#!/bin/ksh
##
## Copyright 2005 IBM Corporation. All rights reserved. This script
## is provided solely as an example for migration planning purposes;
## no warranty, expressed or implied, is provided for this utility.
##
# Script to migrate user information
# The script does the following:
#
# i. adds the user
# ii. displays user information in LDAP
# iii. sets the user's temporary password
# iv. expires the user's password, forcing a change at first log on
#
export AUTH=KRB5LDAP
export PASSWD=tempONEtemp

# if the admusr user does not exist, create it.
lsuser -R $AUTH admusr || /usr/bin/mkuser -R $AUTH -a account_locked=true
admin=true admusr

# extract the username, UID and gecos information from the input file
cat /mnt/user_data.out | awk -F":" '
# add user routine with UID & gecos info
adduser=sprintf("/usr/bin/mkuser -R $AUTH -a id=%s gecos=\"%s\" %s", $2, $3,
$1)
# display user information from LDAP routine
lsuser=sprintf("/usr/sbin/lsuser -R $AUTH %s", $1)

# set the the temporary password routine
passwd=sprintf("/usr/krb5/sbin/kadmin.local -q \"change_password -pw $PASSWD
%s\
\"", $1)

# expire the temporary password routine
chuser=sprintf("/usr/bin/chuser -R $AUTH krb5_attributes=+needchange %s", $1)
# run the routines
system(adduser)
system(lsuser)
system(passwd)
system(chuser)
# single quote that follows ends the awk statement...
,
```

AFS to Kerberos/LDAP group migration

Example E-6 provides a script to migrate groups from AFS to Kerberos/LDAP.

Example: E-6 Script migrate_afs_groups.pl to migrate groups: AFS to Kerberos/LDAP

```
#!/usr/bin/perl
##
## Copyright 2005 IBM Corporation. All rights reserved. This script
## is provided solely as an example for migration planning purposes;
## no warranty, expressed or implied, is provided for this utility.
##
##
## Usage: perl migrate_afs_groups.pl --file <file-name>
##

use Getopt::Long qw(GetOptions);

my $fileName = "";
my $admin_user = "admsvr";

GetOptions ( 'file=s' => \$fileName);

die "Usage: migrate_afs_groups.pl --file <pts_file_name>\n" if (!$fileName);

open (GROUP_IN, "$fileName") ||
    die "failed to open input file $fileName: $!\n";

while (my $data = <GROUP_IN>) {

    next if ($data =~ /^Name /); ## skip first line of file (header)
    next if ($data =~ /^system:); ## skip all AFS "system:" groups

    $data =~ s/\s+/ /g; ## remove multiple spaces

    my ($item1, $item2, $item3, $item4) = split / /, $data;

    next if ($item3 =~ /^system/);

    $item2 =~ s/-//; ## remove "-" from GID

    my ($owner, $user) = "";

    if ($item1 =~ /\:/) {
        ($owner, $user) = split /\:/, $item1;
        $item1 = $user;
    }

    if ($item3 =~ /\:/) {
```

```

        ($owner, $user) = split /\:/, $item1;
        $item3 = $owner;
    }

    ## special case: $item3 is the admin user

    if ($item3 ne "$admin_user") {
        print "creating group $item1, owner $item3. $admin_user is also an
admin\n";
        system("mkgroup -R KRB5LDAP adms=$admin_user,$item3 $item1");
    } else {
        print "creating group: $item1\n";
        system("mkgroup -R KRB5LDAP -a $item1");
    }
}

exit(0);

```

AFS to NFSv4 ACL migration

Example E-7 provides a script to migrate AFS ACLs to NFSv4.

Example: E-7 Script to migrate AFS ACLs to NFSv4

```

#!/usr/bin/perl

## migacl.pl -- Example ACL migration script
## to show the steps required to copy existing AFS ACL information
## into a format usable by NFSv4 'aclput' commands.
##
## Copyright 2005 IBM Corporation. All rights reserved. This script
## is provided solely as an example for migration planning purposes;
## no warranty, expressed or implied, is provided for this utility.
##
## Theory of Operation.
## The script uses standard methods to recursively read a user's
## directory tree, executing "fs listacl" commands on each
## file and sub-directory and using the results to generate NFSv4-
## style ACL input files in the directory $acl_directory. At the
## same time it generates a shell script, <cwd>/acl_<user_id>.sh,
## containing a series of aclput -i commands referencing the files
## found in $acl_directory and the fully qualified file path, e.g.
## "aclput -i <cwd>/$acl_directory/filename.txt.acl <cwd>/filename.txt"
##
## Assumptions.
##

```

```

## The base cell name (obtained using the file /usr/vice/etc/ThisCell) is
## trimmed from the CWD in order to restructure the namespace;
## if the current cell is ../../test.itsc.austin.ibm.com, then
## this string, along with the "/fs" AFS root, will be removed
## from all path names. The --prefix switch can be used to alter
## the new root directory to /nfs/ or any other value (the default
## is / if the --prefix switch is no specified).
##
## We assume the directory structure has been copied intact from
## the AFS cell to the new NFSv4 environment; deviations from this
## basic methodology can be handled by manually editing the output
## shell script or altering the base Perl code.
##

use Cwd; # module for finding the current working directory
use Getopt::Long qw(GetOptions);

my $user = $ENV{"LOGIN"};
my $home = $ENV{"HOME"};
my $main_group = $ENV{"GROUP"};
my $parsed_home = "";
my $acl_directory = "";
my $acl_base = "acl_data";
my $output_file = "";
my ($login,$pass,$uid,$gid) = getpwnam($user);
my $group = getgrgid($gid);
my $do_symlinks = ''; ## by default we'll ignore symlinks
my $prefix = ""; ## default
my $cell_name = "/...";

$|=1;    # turn off I/O buffering

sub ScanDirectory {
    my ($workdir, $output_file, $prefix, $cell_name) = @_;

    ## we should ignore both the "acl_base" directory and OldFiles
    return if ($workdir eq $acl_base);
    return if ($workdir eq "OldFiles");

    my($startdir) = &cwd; # keep track of where we began

    chdir($workdir) or die "Unable to enter dir $workdir:!\n";

    opendir(DIR, ".") or die "Unable to open $workdir:!\n";
    my @names = readdir(DIR);
    closedir(DIR);

    foreach my $name (@names){
        next if ($name eq ".");

```

```

next if ($name eq "..");

if (-d $name){ # is this a directory?
    &ScanDirectory($name, $output_file, $prefix, $cell_name);
    next;
}
unless (&CheckFile($name, $output_file, $prefix, $cell_name)){
    print &cwd."/". $name."\n"; # print the bad filename
}
}
chdir($startdir) or die "Unable to change to dir $startdir: $!\n";
}

sub CheckFile{
    my($name, $output_file, $prefix, $cell_name) = @_;

    ## use the FQ path when checking the existing file
    my $file_to_check = &cwd."/". $name;

    ## now generate a new path to this subdirectory once it's been
    ## migrated to NFSv4.
    my $current_dir = &cwd;
    $current_dir =~ s/$cell_name//;
    my $destination_file = $prefix."/". $current_dir."/". $name;
    $destination_file =~ s/\//\\/g; ## remove doubled "/"

    ## If this file is a symbolic link and we we turned off processing
    ## of these files, then we'll just return from this function.
    return if ( (-l $file_to_check) && ($do_symlinks eq '') );
    print "Traversing symlink: $file_to_check\n" if (-l $file_to_check);

    ## attempt to read the directory entry for this file
    print STDERR "Scanning $file_to_check\n";

    ## Try to open the file. If we fail, we'll just drop out of the
    ## loop and try the next one. Sites should decide if this is
    ## what they want to do, or if they prefer other processing to
    ## occur in such a case.
    if (open (ACL_DATA, "/usr/afs/bin/fs listacl $file_to_check 2>&1 |") ) {

        ## First we'll open a file in which to write NFSv4 ACL commands,
        ## then we'll read data a line at a time from the return
        open (ACL_FILE, ">$acl_directory/$name.acl") ||
            die "failed to open output $acl_directory/$name.acl: $!\n";

        print ACL_FILE "*\n* ACL_type    NFS4\n*\n*\n* Owner: $user\n* Group:
        $group\n*\n";
        ## Next we write our aclput command to our master script.
        print OUTFILE "aclput -i $new_acl_dir/$name.acl $destination_file\n";
    }
}

```



```

## now we loop through returned data to parse the DCE ACLs into
## a form usable by the NFSv4 'aclput' utility. There are two
## possible results: {user|group|other|mask}_obj + permissions, or
## {user|group} + user/group name + permissions. The former will
## be written to the file using a type::acl format, while the rest
## need to appear as type:id:acl, where "id" is either user or
## group.

my @base_perms = @ext_perms = ();
while (my $data = <ACL_DATA>) {

    ## ignore lines starting with "Access" or "Normal"
    next if ($data =~ /^Access list/) || ($data =~ /^Normal/);

    my ($user_id, $perm_mask) = split " ", $data;

    ## If this is an "owned" group, we need to re-parse the
    ## real group and remove the owner.
    if ($user_id =~ /:/)
    {
        my ($owner, $group) = split /:/, $user_id;
        $user_id = $group;
    }

    next if ($user_id eq "authuser"); ## discard this group

    ## we need the ability to work on discrete permission bits;
    ## to do this, we'll transform the permissions list into
    ## an array.

    my ($bit1, $bit2, $bit3, $bit4, $bit5, $bit6, $bit7) =
        split "", $perm_mask;
    my @perms = ($bit1, $bit2, $bit3, $bit4, $bit5, $bit6, $bit7);

    ## next step is the actual conversion of an AFS ACL to an
    ## NFSv4 equivalent. In some cases, a single bit may need
    ## to become a 2- or even 3-bit equivalents in NFSv4.

    ## Important: this example script will not handle all the
    ## possible ACL permutations and only creates "allow" (a)
    ## ACL entries. Further development is left as an exercise
    ## for individual sites requiring a customized solution.

    my $acl_string = "";

    for (my $i=0; $i<=6;$i++) {
        my $temp = "";
        $temp = "r" if ($perms[$i] eq "r" && $acl_string !~ /r/);
    }

```

```

$temp = "w" if ($perms[$i] eq "w");
$temp = "w" if ($perms[$i] eq "k" && $acl_string !~ /w/);
$temp = "C" if ($perms[$i] eq "a");
$temp = "dD" if ($perms[$i] eq "d");
if ($perms[$i] eq "l") {
    if ($acl_string !~ /r/) {
        $temp = "racx"
    } else { $temp = "acx"; }
}
if ($perms[$i] eq "i") {
    if ($acl_string !~ /w/) {
        $temp = "wp"
    } else { $temp = "p"; }
}

$acl_string = "$acl_string$temp";
}
## Next, we start populating arrays that will later be used
## to add the data to our ACL input file. This is extremely
## cumbersome since ACL output from the AFS fs listacl
## command does not distinguish a group from a user ID; thus
## we'll verify each identity using getpwnam. If we get a
## match, the user_id is in the passwd file (therefore a
## user, not a group). Else, it's a group.

push @base_perms, "s:(OWNER@): a $acl_string\n"
if ($user_id eq $user);

push @base_perms, "s:(GROUP@): a $acl_string\n"
if ($user_id eq $main_group);

push @base_perms, "s:(EVERYONE@): a $acl_string\n"
if ($user_id eq "anyuser");

## Start group processing here
if (!getpwnam($user_id) || ($data =~ /:/)) {
    push @ext_perms, "g:$user_id: a $acl_string\n";
} else {
    push @ext_perms, "u:$user_id: a $acl_string\n";
}

} ## end of while loop

my $line = "";
## now, write all the data to the output file...we need to write
## the ext_perms (containing those acls that aren't in the
## "special who" category) first to ensure they're evaluated
## first in the file.

```

```

        print ACL_FILE $line while ($line = pop(@ext_perms));
        print ACL_FILE $line while ($line = pop(@base_perms));

        close ACL_FILE;
        close ACL_DATA || warn "error: $! $?";
    }

    return 1;
}

sub print_help
{
    print "Usage: perl migacl.pl [--help] [--symlinks]\n";
    print "--prefix=<new_namespace_prefix>\n";
    print "    --symlinks: the script will traverse symlinks (default: off)\n";
    print "    --prefix: specifies the top-level directory prefix in use in\n";
    print "                the NFSv4 namespace. This will replace the existing AFS\n";
    print "                \"/.../existing.cell.name/fs\" prefix (default: /) \n";
    print "    --help: prints this help text\n";
    exit(0);
}

## main logic

GetOptions ( 'symlinks' => \$do_symlinks,
             'help' => \$print_help,
             'prefix=s' => \$prefix);

&print_help() if ($print_help);

## Get the current AFS cell name
if (open (CELL_DATA, "/usr/vice/etc/ThisCell") ) {
    my $data = <CELL_DATA>; ## read only 1 line
    $cell_name = $data;
} else {
    print "Error retrieving cell name: $!\n";
    exit(1);
}
close CELL_DATA;

$cell_name = "/afs/$cell_name"; ## prepend /afs prefix

$parsed_home = $home; ## need both
$parsed_home =~ s/$cell_name//;

## Only prepend a prefix if it's non-null.

```

```

$parsed_home = "$prefix/$parsed_home" if ($prefix ne "");

## New base directory
$parsed_home =~ s/\/\//g; ## remove any doubled "/"

## location of .sh file in current cell
$output_file = "./acl_$user.sh"; ## start at CWD in existing cell

## Set up "destination" directories
$acl_directory = $home."/".$acl_base;
$new_acl_dir = $parsed_home."/".$acl_base;

## Emit our expected parameters
print "Starting at base directory: $home.\nDestination directory:
$parsed_home\n";

## location for acl output files in current cell
system("mkdir $acl_directory") if (!-e "$acl_directory");

open (OUTFILE, ">$output_file");
print OUTFILE "#!/bin/sh\n";

&ScanDirectory(".", $output_file, $prefix, $cell_name);

close OUTFILE;
exit(0);

```

Migrate DCE groups to LDAP

Example E-8 provides a script to generate **mkgroup** commands using DCE group data.

Example: E-8 Script to generate mkgroup commands using DCE group data

```

#!/usr/bin/perl

## migrate_dce_groups_to_ldap.pl -- extracts group data from the DCE
## registry using dcecp commands, then uses it to write a shell
## script containing appropriate mkgroup syntax to re-create the
## same groups and GIDs in LDAP.
##
## Copyright 2005 IBM Corporation. All rights reserved. This script
## is provided solely as an example for migration planning purposes;
## no warranty, expressed or implied, is provided for this utility.
##
## Theory of operation: the script creates an output file into
## which it writes individual mkgroup commands containing data

```

```

## extracted from the registry using 1) dcecp -c group cat and 2)
## dcecp -c group show <group_id>. The former command is used
## to obtain the actual group name, while the latter provides
## the GID mapping. For ease of use, each new group in the
## KRB5LDAP registry is owned by the user running the script; thus
## it is recommended that the script be run by an administrative
## user such as root.
##
## Additional "group administrator" data is not extracted from
## the DCE registry, thus only the user running the script will
## have administrative control in the new environment (additional
## work capturing ACLs from the DCE registry would increase the
## options in this area.)
##

use warnings;
use strict;

my $output_file = "./dce_exported_groups.sh";
my $user_id = $ENV{"USER"};
my ($grp, $grp_id, $grp_name) = "";
my $adm_user = "grpadm";

## main logic

## GetOptions ( 'user=s' => \$user_principal);

## First, make sure we can write our output somewhere
if (open (OUTPUT_FILE, ">$output_file")) {
    print OUTPUT_FILE "#!/bin/sh\n\n";

    print OUTPUT_FILE "mkuser -R KRB5LDAP $adm_user\n"; ## group admin

    if (open (GROUP_CAT, "dcecp -c group cat -simple 2>&1 |") ) {
        while ($grp_name = <GROUP_CAT> ) {
            ## skip foreign groups or internal "subsys/dce" entries
            next if ($grp_name =~ /\^\/.../ || $grp_name =~ /\^subsys/ ||
                $grp_name =~ /\^\/\^/);
            chomp $grp_name;

            ## Now we do a "group show" of the returned group
            if (open (GROUP_SHOW, "dcecp -c group show $grp_name 2>&1 |") ){
                while (my $grp_info = <GROUP_SHOW>) {
                    $grp_info =~ s/\^{\|}$/g;
                    ($grp, $grp_id) = split / /,$grp_info if ($grp_info =~ /gid/);
                    chomp $grp_id if ($grp_id);
                }
            }
        }
    }
}

```

```

    }
} else {

    ## here we throw an error but continue nonetheless

    print "Error retrieving data for group $grp_name: $!\n";
}
close GROUP_SHOW;

## here we write the command to generate a new group in the NFS
## space. We preserve the GID but do not retrieve any administrative
## information (e.g. dcecp -c acl show ./:/sec/group/<group_name>)
## as part of this example. This work is left as an exercise for
## the reader should preservation of administrative group access
## on a per-user basis be necessary.

    print OUTPUT_FILE "mkgroup -RKR5LDAP id=$grp_id adms=$user_id
$grp_name\n";
    ($grp_id, $grp_name) = "";
}
close GROUP_CAT;
}

} else {
    print "Error opening output file $output_file: $!\n";
    exit(1);
}

close OUTPUT_FILE;
exit(0);

```

Migrate DCE groups to LDAP

Example E-9 provides a script to generate **mkuser** commands using DCE user account data.

Example: E-9 Script to generate mkuser commands using DCE user account data

```

#!/usr/bin/perl

## migrate_dce_users_to_ldap.pl -- extracts user accounts from a DCE
## registry using dcecp commands, then uses it to write a shell
## script containing appropriate mkuser syntax to re-create the
## users in the KRB5LDAP database.
##
## Copyright 2005 IBM Corporation. All rights reserved. This script
## is provided solely as an example for migration planning purposes;

```

```

## no warranty, expressed or implied, is provided for this utility.
##
## Theory of operation: the script creates an output file into
## which it writes individual mkuser commands containing data
## extracted from the registry using 1) dcecp -c account catalog
## and 2) dcecp -c user show <user_id>. The former command is used
## to generate a list of users, while the second retrieves discrete
## account settings for each user. The script can also be run
## with the --user <user_id> option in order to retrieve data for
## one account.
##
## Assumptions:
##
## The script uses the dcecp -c getcellname command to return the
## DCE cell name; it does this in order to strip off this string
## from, for instance, the {home <home directory>} line in the
## output from the user show command. We presume a site will
## wish to remove this prefix and, optionally, replace it with
## another (e.g. /nfs/home). Thus if a user's home directory
## in DFS is ../../cell_name.ibm.com/fs/u/j/jones, and the script
## is run with --prefix /nfs/home, the resulting home directory
## syntax will be /nfs/home/u/j/jones. Without the prefix
## flag, it would be /u/j/jones.
##
## It is also presumed the DCE "none" group will be abandoned
## in the new environment. The script replaces this with a new
## default, which is "migusr" unless $default_group is changed
## prior to execution. Specify --nodefault on the command line
## to prevent a "mkgroup $default_group" from being written to
## the output shell script.
##
## Note that passwords cannot be migrated using this method, as
## it is not possible to decrypt the string stored in the DCE
## registry.
##

use strict;
use warnings;

use Getopt::Long qw(GetOptions);

my $output_file = "./dce_exported_users.sh";
my ($user_principal, $insert_string, $default_yes) = "";
my $prefix = "/";
my $default_group = "migusr"; ## replacement for DCE's "none"
my ($item, $uid, $group, $home, $shell, $fullname, $cell_name) = "";
my @user_list = ();
my @group_list = ();

```

```

## main logic

GetOptions ( 'user=s' => \$user_principal,
             'prefix=s' => \$prefix,
             'nodefault' => \$default_yes);

## Get the current DCE cell name
if (open (CELL_DATA, "dcecp -c getcellname 2>&1 |") ) {
    my $data = <CELL_DATA>; ## read only 1 line
    $cell_name = $data;
} else {
    print "Error retrieving cell name: $!\n";
    exit(1);
}
close CELL_DATA;

$cell_name =~ s/\n/\fs/; ## strip carriage return and add DFS /fs suffix

open (OUTPUT_FILE, ">$output_file") || die "unable to open output file: $!\n";

print OUTPUT_FILE "#!/bin/sh\n\n";

## create the default group. If this is not desirable, or if
## a preferred default group was created when the group-
## migration script was run, simply specify --nodefault on
## the command line.

print OUTPUT_FILE "mkgroup -R KRB5LDAP $default_group\n" if (!$default_yes);

## One user only
if ($user_principal) {
    push @user_list, $user_principal;
    print "Processing user $user_principal\n";
} ## all users
} else {
    if (open (USER_DATA, "dcecp -c account cat -simple 2>&1 |") ) {
        while (my $user_id = <USER_DATA>) {
            next if ($user_id =~ /\//); ## bypass all accts with "/" in name
            chomp $user_id;
            push @user_list, $user_id;
        }
    } else {
        print "Error generating account list: $!\n";
        exit(1);
    }
}

close USER_DATA;

```



```

while (my $user_principal = pop(@user_list)) {

    print "Generating command for $user_principal\n";
    if (open (USER_DATA, "dcecp -c user show $user_principal 2>&1 |") ) {

        while (my $input_line = <USER_DATA> ) {

            chomp $input_line;
            $input_line =~ s/^\{|\}$//g;
            ($item, $uid) = split " ", $input_line if ($input_line =~ "^uid");

            if ($input_line =~ "^groups") {
                $input_line =~ s/^groups /^group:/;
                ($item, $group) = split ":", $input_line;
            }
            ($item, $home) = split " ", $input_line if ($input_line =~ "^home");
            ($item, $shell) = split " ", $input_line if ($input_line =~ "^shell");
            ($item, $fullname) = split "{", $input_line
                if ($input_line =~ "^fullname");
        }

        ## post-process certain entries

        $insert_string = "uid=$uid";

        $fullname =~ s/)//; ## remove remaining }

        $insert_string = "$insert_string gecos=\"$fullname\" " if ($fullname);

        ## special handling for groups.  If the first listed group
        ## is "none" then we'll replace it with the value of the
        ## $default_user variable.

        @group_list = split / /, $group ;
        if ($group_list[0] eq "none") {
            $insert_string = "$insert_string pgrp=$default_group";
        } else {
            $insert_string = "$insert_string pgrp=$group_list[0]";
        }

        $group="";
        while (my $grp = pop @group_list) {
            next if ($grp =~ /\//); ## remove groups with "/" chars
            $group = "$group,$grp" if ($grp ne "none"); ## ignore DCE "none"
        }
        $group =~ s/^\,//; ## we can end up with a leading comma

        $insert_string = "$insert_string groups=$group" if ($group);
    }
}

```

```

    ## clean up the $HOME, removing the old DCE prefix
    $home =~ s/$cell_name/$prefix/;
    $home =~ s/\\\/\\\/g;

    ## print "Name: $fullname\nUID: $uid\nGroup: $group\nHome: $home\nShell:
    $shell\n";
    $insert_string="$insert_string home=$home" if ($home ne "/");

    } else {
        print "Error parsing command for $user_principal\n";
        next;
    }
    close USER_DATA;

    print OUTPUT_FILE "mkuser -R KRB5LDAP $insert_string $user_principal\n";
}

close OUTPUT_FILE;

exit(0);

```

Copy ACL

Example E-10 provides a script to copy an ACL (with recursive option).

Example: E-10 Script to copy an ACL (with recursive option)

```

#!/usr/bin/ksh
##
## Copyright 2005 IBM Corporation. All rights reserved. This script
## is provided solely as an example for migration planning purposes;
## no warranty, expressed or implied, is provided for this utility.
##
# copy_acl.sh
#
# Copy the ACL for the given source file/directory to other files/directories
#

# Name of this script
scrname=${0##*/}

#
# Functions
#

```

```

function usage {
    echo "Usage: $scrname [-R] <source> <dest>"
    echo "  where"
    echo "  -R indicates a recursive copy"
    echo "      (copy ACL to all files and directories below and including"
    echo "      the destination.)"
    echo "  <source> = the name of the file or directory to copy the ACL from"
    echo "  <dest>   = the name of the file or directory to copy the ACL to"

    exit 1
}

if [[ $# -eq 0 ]]
then
    usage
fi

#
# Process input parameters
#

if [[ "$1" = "-R" ]]; then
    SETSUBTREE="true"
    shift
else
    SETSUBTREE="false"
fi

if [[ -n "$1" ]]; then
    SRC_NAME="$1"
else
    usage
fi

if [[ -n "$2" ]]; then
    DEST_NAME="$2"
else
    usage
fi

#
# Initialize other variables
#

NBERR=0
TMP_ACLFILE="/tmp/.AIXACL_$$"

if [[ -e "${SRC_NAME}" ]]; then
    aclget -o "${TMP_ACLFILE}" "${SRC_NAME}"

```

```

        NBERR=$?
    else
        echo "Source \"${SRC_NAME}\" does not exist"
        NBERR=1
    fi

    if [[ "${NBERR}" -eq 0 ]]; then
        if [[ -e "${DEST_NAME}" ]]; then
            if [[ -d "${DEST_NAME}" && "${SETSUBTREE}" = "true" ]]; then
                find "${DEST_NAME}" -print | while read NAME
                do
                    aclput -i "${TMP_ACLFILE}" "${NAME}"
                    (( NBERR += $? ))
                    ls -dl "${NAME}"
                done
            else
                aclput -i "${TMP_ACLFILE}" "${DEST_NAME}"
                (( NBERR += $? ))
                ls -dl "${DEST_NAME}"
            fi
        else
            echo "Destination \"${DEST_NAME}\" does not exist"
            NBERR=1
        fi
    fi

    rm -f "${TMP_ACLFILE}"
    exit ${NBERR}

```

Installing an AIX 5L maintenance level

The appendix describes the basic steps required to install a new maintenance level on AIX 5L V5.3.

Obtaining the latest fixes

In this section, we describe different ways in which to obtain the latest fixes.

On the Web

You can obtain the AIX 5L 5300-03 Recommended Maintenance package from the Maintenance packages section of Quick links for AIX fixes:

<http://www.ibm.com/servers/eserver/support/unixservers/aixfixes.html>

This site also has instructions about how to install the recommended maintenance package.

AIX 10/2005 Update CD

An Update CD is shipped with all new orders of AIX 5L V5.3. The 5300-03 Recommended Maintenance package is included on the 10/2005 and later Update CDs. Existing pSeries customers who are licensees of AIX 5L V5.3 can obtain the Update CD at no charge, except for media charges as they apply in their geography, by contacting their point of sale and requesting APAR number IY71011 or Feature Code 0970.

Installation tips

When installing the maintenance package, consider the following tips:

- ▶ You need to be logged in as root to perform the installation of this package.
- ▶ We recommend that you create a system backup before starting the installation procedure. Refer to the **mksysb** command in the *AIX 5L V5.3 Commands Reference* for additional information.
- ▶ The latest AIX 5L V5.3 installation hints and tips are available from the Subscription service for UNIX servers Web page at:

<https://techsupport.services.ibm.com/server/pseries.subscriptionSvc>

These tips contain important information that should be reviewed before installing this update.

Installation

To install selected updates from this package, use the following command:

```
smitty update_by_fix
```

To install all updates from this package that apply to installed filesets on your system, use the following command:

```
smitty update_all
```

We highly recommend that you install all the updates from this package.

After a successful installation, a system reboot is required for this update to take effect.

Example F-1 shows a sample menu screen from the **smitty update_all** command.

Example: F-1 Sample menu screen from the smitty update_all command

```
Update Installed Software to Latest Level (Update All)
```

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

	[Entry Fields]
* INPUT device / directory for software	/dev/cd0
* SOFTWARE to update	_update_all
PREVIEW only? (update operation will NOT occur)	no +
COMMIT software updates?	no +
SAVE replaced files?	no +
AUTOMATICALLY install requisite software?	yes+
EXTEND file systems if space needed?	yes+
VERIFY install and check file sizes?	no +
DETAILED output?	no +
Process multiple volumes?	yes+
ACCEPT new license agreements?	yes+
Preview new LICENSE agreements?	no +

We recommend that you do not COMMIT filesets. If you COMMIT the filesets, you will not be able to uninstall the Recommended Maintenance Level (RML) if problems are encountered. The RML installation can always be COMMITted after you are satisfied that your system is functioning as you expected. You can run the following SMIT command to COMMIT any APPLIED filesets:

```
smitty commit
```

Verifying the installation

To determine if your system is running AIX 5L V5.3 RML03, run the commands shown in Example F-2 on page 388 on your system or systems.

Example: F-2 Confirming the version and RML of AIX 5L installed on a system

```
# lppchk -v
#
# oslevel -r
5300-03
#
```

Example F-2 shows that the version of AIX 5L installed on our system is AIX 5L V5.3 RML03. If the output from the **oslevel -r** command shows an earlier version of AIX 5L, you are either not running the required level of AIX 5L or RML03 has not been correctly installed. Use the **lppchk -v** command to check what fileset had problems installing.

If the **lppchk -v** command returns no output and the **oslevel -r** command shows a RML version earlier than what you just installed, run the command shown in Example F-3.

Example: F-3 Using the oslevel command to determine down-level filesets

```
# oslevel -rl 5300-03
Fileset                                Actual Level      Recommended ML
-----
devices.pci.14109f00.rte              5.3.0.0           5.3.0.30
#
```

The output in Example F-3 indicates that the `devices.pci.14109f00.rte` fileset installed on the system is at Version 5.3.0.0 and the system is expecting it to be at Version 5.3.0.30. You will need to install the expected version of the fileset to bring the system to RML03.

Sample migration planning worksheet

When performing any major migration to a new environment, thorough planning is the key to success. We provide the form in Figure G-1 on page 390 as an example of the considerations that might be required for any basic NFSv4 migration. Production environments will, of course, have additional considerations for customized solutions and applications that should be added to any migration plan.

Sample NFSv4 Migration Worksheet

Authentication (ID Migration)

Current authentication method: _____

New authentication method: _____

What (if any) intermediary steps are required for conversion? _____

- 1) _____
- 2) _____
- 3) _____

Database conversion method: _____

ACL Migration

Current ACL type: _____

New filesystem type: (JFS2 or GPFS) _____

Are NFSv4 ACLs required? _____

Special ACL conversion required? _____

Conversion steps:

- 1) _____
- 2) _____
- 3) _____

Data Migration

New name space: _____

Migration type: _____

Data move steps:

- 1) _____
- 2) _____
- 3) _____

ACL Restore Method: _____

Figure G-1 Sample migration worksheet

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described in this appendix.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246657>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select **Additional materials** and open the directory that corresponds with the redbook form number, SG246657.

Using the Web material

The additional Web material that accompanies this redbook includes the following file:

<i>File name</i>	<i>Description</i>
SG246657_Migrate.ZIP	Migration example Perl scripts from Appendix E, "Scripts and configuration files" on page 353

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material ZIP file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 395. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Securing NFS in AIX: An Introduction to NFS V4 in AIX 5L Version 5.3*, SG24-7204
- ▶ *AIX - Migrating NIS Maps into LDAP*, TIPS-0123
- ▶ *AIX 5L Differences Guide Version 5.3 Edition*, SG24-7463
- ▶ *AIX and Linux Interoperability*, SG24-6622
- ▶ *Understanding LDAP- Design and Implementation*, SG24-4986
- ▶ *DCE Replacement Strategies*, SG24-6935
- ▶ *IBM's General Parallel File System (GPFS) 1.4 for AIX*, REDP-0442

Other publications

These publications are also relevant as further information sources:

- ▶ *General Parallel File System (GPFS) for Clusters: Administration and Programming Reference*, SA22-7967
- ▶ *AIX 5L Version 5.3 Security Guide*, SC23-4907
- ▶ *AIX 5L Version 5.2 Performance Management Guide*, SC23-4876
- ▶ *AIX 5L Version 5.3 System Management Guide: Communications and Networks*, SC23-4909
- ▶ *AIX 5L Version 5.3 Files Reference*, SC23-4895
- ▶ *AIX 5L Version 5.3 System User's Guide: Communications and Networks*, SC23-4912

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ AIX 5L V5.3 online documentation
<http://publib.boulder.ibm.com/infocenter/pseries/index.jsp>
- ▶ AIX 5L V5.2 online documentation
http://www16.boulder.ibm.com/pseries/en_US/infocenter/base/aix52.htm
- ▶ IBM @server pSeries and AIX information center
http://www16.boulder.ibm.com/pseries/en_US/infocenter/base/index.htm
- ▶ Quick links for AIX fixes
<http://www.ibm.com/servers/eserver/support/unixservers/aixfixes.html>
- ▶ AIX 5L Version 5.3 Commands Reference
<http://publib.boulder.ibm.com/infocenter/pseries/index.jsp>
- ▶ IBM @server Cluster information center
<http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp>
- ▶ IBM AFS Support
<http://www.ibm.com/software/stormgmt/afs/library/>
- ▶ IBM @server Fix Central
<http://www.ibm.com/eserver/support/fixes/>
- ▶ IBM Tivoli Directory Server
<http://www.ibm.com/software/tivoli/products/directory-server>
- ▶ Subscription service for UNIX servers
<https://techsupport.services.ibm.com/server/pseries.subscriptionSvc>
- ▶ *IBM Distributed Computing Environment Version 3.2 for AIX and Solaris: DCE Security Registry and LDAP Integration Guide*
<http://www.ibm.com/software/network/dce/library/publications/ldaprgy/html/LDAPRG02.HTM>
- ▶ *Improving Database Performance With AIX Concurrent I/O: A case study with Oracle9i Database on AIX 5L version 5.2* white paper
http://www.ibm.com/servers/aix/whitepapers/db_perf_aix.pdf
- ▶ NFSv4: General Information and References for the NFSv4 protocol
<http://www.nfsv4.org>
- ▶ Pawlowski, B. et al., *The NFS Version 4 Protocol*
<http://www.nluug.nl/events/sane2000/papers/pawlowski.pdf>

- Comprehensive Perl Archive Network (CPAN)

<http://www.cpan.org>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols

/etc/exports 25, 32, 41, 182, 340
/etc/filesystems 340
/etc/irs.conf file 86
/etc/nfs/hostkey 340
/etc/nfs/local_domain 340
/etc/nfs/princmap 340
/etc/nfs/realm.map 340
/etc/rmtab 340
/etc/sm 340
/etc/sm.bak 340
/etc/state 340
/etc/syslog.conf 197
/etc/xtab 340

A

aclconvert command 60, 98, 258
acledit command 59–60, 63, 98
aclget command 63, 98, 107
aclgettypes command 98
aclput command 98, 107

ACLs

commands 98
evaluation 112
evaluation flowchart 113
flowchart 113
inheritance 62
JFS2 59
migrating 176
types
 NFSv4 ACL 90

AFS

ACL 279
caching 266
cell 262
file system semantics 272
location transparency 264
NFS/AFS Translator 278
replication 266
security 266
volumes 265

AFS to NFS migration

ACLs 280

example 300
cell to domain 288
data migration 276
 example 290
groups 270
 example 293
namespace 273
security 268
 example 290
users 269
 example 290
AFS versus NFS
 ACLs 280
 cell and domain 272
 namespace 273
 semantics 273
aio 125
AIX 5L
 RML03 26
AIX 5L V5.3 26
 RML03
 installation 386
authentication 170, 173
 host 181
 methods 178
authorization
 exports 182
 host 182
automating replication
 (using rdist) 49
automount command 341

B

biod daemon 340

C

cache
 consistency 28
cache file system
 configuring 84
 performance 83
cache file system (see CacheFS) 82
CacheFS 27, 82

- benefits 82
- configuring 84
- performance 82–83
- caching 13
- CDS 179
- Cell Directory Services 179
- cfsadmin command 84–85
- chmod command 176
- chnfs command 41, 208, 342
- chnfsdom command 342
- chnfsexp command 341
- chnfsim command 342
- chnfsmnt command 341
- chnfsrtd command 209
- chnfssec command 342
- CIO (see concurrent I/O) 57
- commands
 - aclconvert 60, 98, 258
 - acledit 59–60, 63, 98
 - aclget 63, 98, 107
 - aclgettypes 98
 - aclput 98, 107
 - automount 341
 - cfsadmin 84–85
 - chmod 102, 176
 - chnfs 41, 208, 342
 - chnfsdom 342
 - chnfsexp 341
 - chnfsim 342
 - chnfsmnt 341
 - chnfsrtd 209
 - chnfssec 342
 - cpio 44–45, 80
 - crontab 49
 - df 32
 - exportfs 32, 34, 182, 341–342
 - kinit 210
 - klist 210
 - ktutil 210
 - ldapadd 132
 - ldapmodify 87
 - lppchk 388
 - lsnfsexp 341
 - lsnfsmnt 341
 - lsuser 255
 - mkcfsmnt 84
 - mkgroup 251
 - mknfs 341
 - mknfsexp 342

- mknfsmnt 342
- mksysb 386
- mkuser 250
- mount 32, 214, 341, 343
- nfs.clean 221
- nfs4cl 42–43, 342, 347
 - options
 - showstat 33
- nfsd 41
- nfshostkey 208, 342
- nfso 28, 341, 344
 - options
 - client_delegation 28
 - deleg 29
 - server_delegation 29
- nfsstat 29
- nistoldif 86
- oslevel 27, 388
- rc.nfs 340
- rdist 44, 46, 48
- refresh 197
- rmnfs 342
- rmnfsexp 342
- showmount 341
- tar 46, 79
- commands mkuser 254
- concurrent I/O 57
- cpio command 44–45, 80
- crontab command 49

D

- DCE/DFS
 - ACLs 237
 - aggregates and filesets 227
 - CDS
 - Cell Directory Services 179
 - cells 225
 - DCE/DFS principal and group considerations
 - users and groups 229
 - DCE/DFS to NFS migration
 - ACL 237
 - data migration 243
 - delegation
 - caching 27
 - controlling on the client 28
 - controlling on the server 29
 - performance 30
 - statistics 30

- df command 32
- DIO (see direct I/O) 57
- direct I/O 57
- directory services 179

E

- enterprise file systems 5
 - characteristics 12
 - concepts 4
 - operational considerations 16
 - technologies 13
 - AFS 14
 - DFS 15
 - NFS 13
- exname 23–24
- exportfs command 32, 34, 182, 341
 - options 342
 - access 342
 - exname 342
 - nfsroot 343
 - refer 343
 - replicas 343
 - ro 342
 - root 342
 - rw 342
 - sec 182, 343
 - vers 182, 342
- exporting replicas 40
- exports file 32, 35, 182, 340
- exports options
 - sec= 182
 - vers= 182
- external namespace 23

F

- failover 36, 43
- features of NFSv4 50
- file handles 25, 36
- files
 - /etc/exports 182, 340
 - /etc/filesystems 340
 - /etc/nfs/hostkey 340
 - /etc/nfs/local_domain 340
 - /etc/nfs/princmap 340
 - /etc/nfs/realm.map 340
 - /etc/sm 340
 - /etc/sm.bak 340
 - /etc/state 340

- /etc/syslog.conf 197
- /etc/xtab 340
- filesystems file 340
- flowchart
 - NFSv4 ACL evaluation 113
- FSID 25
- fsid 25–26, 36

G

- General Parallel File System (See GPFS) 63
- Global Storage Architecture (See GSA) 314
- GPFS 63
 - advantages 67
 - architecture 65
 - cache usage with NFS 72
 - exporting with NFSv4 72
 - NFSv4 access controls lists
 - changing 77
 - deleting 77
 - displaying 77
 - limitations 78
 - NFSv4 ACL translation 75
 - planning 70
 - usage with GSA 317
 - using with NFSv4 70
- GSA 314
 - backups 322
 - benefits 325
 - hardware 319
 - Kerberos 323
 - software 322
 - status 326
 - time synchronization 323
- gssd daemon 341

I

- IBM Tivoli Directory Server 122
 - configuration 127
 - installation 126
- IBM Tivoli Directory Server (see also LDAP) 122
- identification
 - host 181
- integrity 170

J

- JFS2
 - ACL types 60

- comparing with JFS 56
- converting acls to NFSv4 60
- features 57
- snapshots 57

K

- kadmin command
 - options
 - ktadd 209
- kadmin.local
 - options
 - add_principal 207
 - get_principal 207
- Kerberos
 - host identification 181
 - machine principal 181
 - service principal 181
- kinit command 210
- klist command 210
- ktutil command 210
 - options
 - read_kt 210

L

- LDAP 85
 - automount maps 85
 - client configuration 133
 - commands
 - ldapmodify 87
 - nistoldif 86
 - configuration 127
 - installation 126
 - usage with GSA 318
 - user registry 172
- ldapadd command 132
- ldapmodify command 87
- local_domain file 340
- lppchk command 388
- lsnfsexp command 341
- lsnfsmnt command 341
- lsuser command 255

M

- migration 12
 - components 169
 - considerations 163
 - hardware planning 168

- security
 - sizing 177
 - types of 164
- migration choices 5
- mkcfsmnt command 84
- mkggroup command 251
- mknfs command 341
- mknfsexp command 342
- mknfsmnt command 342
- mksecdap command
 - syntax 133
- mksysb command 386
- mkuser command 250, 254
- mount command 32, 214, 341, 343
 - options
 - acl 344
 - bg 343
 - cio 344
 - dio 344
 - fg 343
 - hard 343
 - intr 343
 - nointr 343
 - ro 343
 - rw 343
 - sec 344
 - soft 343
 - vers 344
 - syntax 216

N

- namespace
 - federated 13
- Network Dispatcher
 - usage with GSA 319
- Network Time Protocol 122
- Network Time Protocol (See NTP) 329
- NFS commands 342
 - automount 341
 - chnfsdom 342
 - chnfsexp 341
 - chnfsim 342
 - chnfsmnt 341
 - chnfssec 342
 - exportfs 341–342
 - lsnfsexp 341
 - lsnfsmnt 341
 - mknfs 341

- mknfsexp 342
- mknfsmnt 342
- mount 341, 343
- nfs4cl 342, 347
- nfshostkey 342
- nfso 341, 344
- rc.nfs 340
- rmnfs 342
- rmnfsexp 342
- showmount 341
- NFS daemons
 - /usr/sbin/biod 340
 - /usr/sbin/gssd 341
 - /usr/sbin/nfsd 340
 - /usr/sbin/nfsrgyd 341
 - /usr/sbin/pcnfsd 341
 - /usr/sbin/portmap 341
 - /usr/sbin/rpc.lockd 340
 - /usr/sbin/rpc.mountd 340
 - /usr/sbin/rpc.statd 340
- NFS registry daemon 341
- nfs.clean command 221
- nfs4cl 26
- nfs4cl command 26, 42–43, 342, 347
 - example
 - resetsoptions
 - example 350
 - options
 - resetsoptions 347
 - setsfoptions 347
 - example 350
 - showfs 349
 - example 349
 - showstat 33
 - example 351
 - showstats 349
- nfsd command 41
 - options
 - getnodes 204
- nfsd daemon 340
- nfshostkey command 208, 342
- nfshostkey file 340
- nfso command 28, 341, 344
 - examples 346
 - NFSv4-specific options
 - client_delegation 346
 - nfs_v4_fail_over_timeout 345
 - nfs_v4_pdts 345
 - nfsv4_vm_bufs 345
 - server_delegation 346
 - utf8 344
 - utf8_validation 344
- nfsrgyd daemon 341
- nfsstat command 29
- NFSv4
 - accounts 171, 173
 - ACLs (see also ACLs - NFSv4 ACLs) 90
 - exports options 182
 - feature list 50
 - host authentication 181
 - host authorization 182
 - host identification 181
 - host principal 207
 - referrals 31
 - replication
 - load balancing 40
 - security 169–170
 - user accounts 171, 173
- NFSv4 ACLs
 - access evaluation 95
 - acedit 98
 - administration 98
 - chmod command 102
 - directory structure 104
 - file system support 91
 - format 91
 - inheritance 94
 - inheritance and move versus copy 104
 - inheritance and umask 103
 - inheritance, maximizing benefits of 105
 - permission bits 92
 - permission restrictions 94
 - permissions scenarios 110
 - special user permissions 94
 - UNIX permissions 97
 - using with JFS2
 - JFS2 ACLs 59
 - with NFSv3 clients 114
- NFSv4 commands
 - nfs4cl 26
- nistoldif command 86
- NTP 122, 329
 - configuration 330

O

- oslevel command 27, 388

P

- pcnfsd daemon 341
- performance 27
- persistent storage 27
- portmap daemon 341
- preferred clients 42
- princmap file 340
- privacy 171
- pseudo root 23, 30, 32

R

- rc.nfs command 340
- rdist command 44, 46, 48
 - automating 49
- realm.map file 340
- Redbooks Web site 395
 - Contact us xvii
- referral 30–31
- referrals 31
- refresh command 197
- replication 12, 31, 36
 - automating 49
 - enabling 34
 - failover 36
 - synchronizing 44
- rmnfs command 342
- rmnfsexp command 342
- rpc.lockd daemon 340
- rpc.mountd daemon 340
- rpc.statd daemon 340

S

- SAN 63
- scripts
 - add users to the KDC 354
 - AFS to Kerberos/LDAP group migration 369
 - AFS to Kerberos/LDAP user migration 368
 - AFS to NFSv4 ACL migration 370
 - DFS to AIXC ACL migration example 256, 356
 - DFS to NFSv4 migration example 361
 - group migration 251
 - mkuser commands 252, 378
 - replica synchronization 49
 - sync replicas 49
 - sync_data.sh 49
- security 169
 - information security 170
 - kerberos 171

- personnel security 170
- physical security 170
- RPCSEC_GSS 170, 172
- showmount command 341
- sizing
 - security migration 177
- sm file 340
- sm.bak file 340
- snapshots 57
- soft mounts 43
- state file 340
- storage area network 63
- synchronizing replicas 44
- syslog.conf file 197
- syslogd daemon 197
 - configuration 197

T

- tar command 46, 79
- test environment 310–311
- timeout
 - setting timeout values 43
- Tivoli Storage Manager 79

U

- user accounts 171, 173
- user authentication 173

W

- Web-based System Manager 99
 - ACL administration 99

X

- X.500 180
- xtab file 340



Implementing NFSv4 in the Enterprise: Planning and Migration Strategies

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Redbooks

Implementing NFSv4 in the Enterprise: Planning and Migration Strategies

Planning and implementation examples for AFS and DFS migrations

NFSv3 to NFSv4 migration examples

NFSv4 updates in AIX 5L Version 5.3 with 5300-03

Recommended Maintenance Package

The most recent maintenance release of IBM AIX 5L Version 5.3 includes a significant set of new features added to the NFSv4 implementation. In 2004, the first IBM Redbook devoted to the topic of NFSv4 implementation in AIX 5L was published: *Securing NFS in AIX: An Introduction to NFS V4 in AIX 5L*, SG24-7204.

This IBM Redbook provides additional up-to-date information to help IBM clients understand and take advantage of the new NFSv4 functions provided by AIX 5L Version 5.3 with the 5300-03 Recommended Maintenance Package.

The NFSv4 implementation in AIX 5L has now expanded to provide core features that make it capable of providing a much broader range of distributed file system services than any prior version of NFS. The scope of this book includes methods for implementing NFSv4 in the enterprise and extensive coverage of methods for how it can potentially be used as a migration target for existing AFS-based and DCE/DFS-based enterprise file systems.

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks