

IBM Tivoli Directory Server for z/OS

Technical overview of Tivoli Directory Server

Concepts, planning, and configuration examples

Basic and advanced replication



Karan Singh
Corey C Bryant
Jonathan Cottrell
Gillian Gainsford
Saheem Granados
Robert Green
Diane Lia
Nilesh T Patel
John M Walsh



International Technical Support Organization

IBM Tivoli Directory Server for z/OS

June 2011

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

First Edition (June 2011)

This edition applies to Version 1, Release 12, of IBM Tivoli Directory Server for z/OS.

© Copyright International Business Machines Corporation 2011. 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

Notices	ix
Trademarks	x
Preface	xi
The team who wrote this book	xi
Now you can become a published author, too!	xii
Comments welcome	xiii
Stay connected to IBM Redbooks	xiii
Part 1. Overview	1
Chapter 1. Tivoli Directory Server for z/OS	3
1.1 z/OS LDAP - features	4
1.2 IBM Tivoli Directory Server for z/OS	4
1.3 Directory Architecture	5
1.4 Server Architectures	5
1.4.1 Single-Server	6
1.4.2 Multi-Server (Sysplex)	8
1.4.3 Master-Replica Replication	10
1.4.4 Forwarding (Cascading) Replication	12
1.4.5 Peer-to-Peer Replication	13
1.4.6 Gateway Replication	15
1.4.7 Remote security services using the z/OS LDAP server	16
1.4.8 Further Information	17
Chapter 2. Planning	19
2.1 Planning and Considerations	20
2.2 Where to store your data	20
2.3 Required products	20
2.3.1 Optional products	21
2.4 Configuring IBM Tivoli Directory Server for z/OS	22
2.4.1 Where and how to store user passwords?	22
2.4.2 Configuring for advanced replication and LDAP password policy	23
2.4.3 Activity and audit logging	23
2.4.4 Using the dsconfig utility	23
Part 2. Concepts	27
Chapter 3. Back ends	29
3.1 Back end overview	30
3.2 TDBM back end	32
3.2.1 TDBM configuration	33
3.2.2 Porting TDBM data from IBM Tivoli Directory Server for z/OS to IBM Tivoli Directory Server for z/OS	34
3.2.3 Porting TDBM data from ISS to IBM Tivoli Directory Server for z/OS	34
3.2.4 Using the TDBM back end	35
3.2.5 Tuning the TDBM back end	37
3.3 LDBM back end	39
3.3.1 LDBM configuration	40

3.3.2	Porting LDBM data	41
3.3.3	Creating a sample server with an LDBM back end	41
3.3.4	Using the LDBM back end	41
3.3.5	Tuning the LDBM back end	41
3.3.6	Sample LDBM benchmark data	43
3.4	CDBM back end	44
3.4.1	CDBM Configuration	44
3.4.2	Using the CDBM back end	46
3.4.3	Tuning the CDBM back end	46
3.5	SDBM back end	46
3.5.1	SDBM Configuration	47
3.5.2	Using the SDBM back end	50
3.5.3	Searching the SDBM back end	52
3.5.4	Tuning the SDBM back end (RACF database)	54
3.5.5	RACF resources	55
3.6	GDBM back end	64
3.6.1	GDBM configuration	66
3.6.2	Enabling change logging	68
3.6.3	Additional configuration for RACF change logging	68
3.6.4	Using the GDBM back end	70
3.6.5	Tuning the GDBM back end	71
Chapter 4.	Schemas	73
4.1	Schema	74
4.2	Schema configuration in IBM Tivoli Directory Server for z/OS	74
4.2.1	Applying schema to IBM Tivoli Directory Server for z/OS	75
4.3	Attribute Types	75
4.3.1	Attributetypes and ibmattributetypes attribute format	79
4.4	Object Classes	81
4.4.1	objectclasses attribute value format	82
4.5	Defining additional schema in IBM Tivoli Directory Server for z/OS	82
4.5.1	Defining additional schema example	83
4.6	Defining additional schema for use with RACF custom fields	87
Chapter 5.	Authentication, authorization, and security	89
5.1	Overview	90
5.2	Authentication mechanisms supported by IBM Tivoli Directory Server for z/OS	92
5.2.1	Anonymous	93
5.2.2	Simple	93
5.2.3	CRAM-MD5	94
5.2.4	DIGEST-MD5	96
5.2.5	GSS-API (Kerberos)	98
5.2.6	External (SSL)	101
5.3	Native authentication	104
5.3.1	Setting up native authentication	106
5.3.2	Changing a password or password phrase of an entry participating in native authentication	107
5.4	Authorization using Tivoli Directory Server Access Control Lists (ACL)	108
5.4.1	Setting up IBM Tivoli Directory Server Authorization	109
5.4.2	Normalization	111
5.4.3	Propagation	112
5.4.4	Authorization Permissions	113
5.4.5	Precedence	114

5.4.6	Determining the Subject	114
5.4.7	Calculating Effective Permissions	115
5.4.8	Filtered Access Control	116
5.4.9	Testing Authorization Configurations	117
5.4.10	Closing thoughts on authorization	120
5.5	Groups and group gathering in IBM Tivoli Directory Server for z/OS	121
5.5.1	Static, dynamic, and nested groups	121
5.5.2	Querying group membership	123
5.5.3	Static, dynamic, and nested group pros and cons	125
5.5.4	Group gathering	126
5.6	Password Policy	127
5.6.1	Multiple password policies	127
5.6.2	Meaning of various attributes in password policy	128
5.7	Encryption and Hashing	132
5.8	SSL/TLS	133
5.8.1	Certificates and key repositories	134
5.8.2	Setting up IBM Tivoli Directory Server for z/OS to use SSL/TLS	135
5.9	Persistent Search	136
Chapter 6. Reliability, availability, and scalability		139
6.1	Reliability, Availability and Scalability	140
6.1.1	Availability	140
6.2	Sysplex	140
6.3	Replication	141
6.4	Topology	142
6.4.1	Master - Replica	143
6.4.2	Peer - Peer	143
6.4.3	Forwarding/Cascading	143
6.4.4	Gateway	143
6.4.5	Sysplex and Replication	144
6.5	Setting up Replication	144
6.5.1	Consumer Configuration	146
6.5.2	Supplier Configuration	146
6.5.3	Synchronizing the servers	149
6.5.4	Maintaining the Topology	150
6.6	Additional Advanced Replication Features	153
6.6.1	Scheduling	153
6.6.2	Filtering	154
Chapter 7. Plug-ins		155
7.1	IBM Tivoli Directory Server for z/OS Server Plug-ins	156
7.2	Pre-operation and post-operation plug-ins	158
7.3	Client-operation plug-ins	160
7.4	Building an IBM Tivoli Directory Server for z/OS server plug-in	162
7.5	Steps for writing a IBM Tivoli Directory Server for z/OS server plug-in	162
7.6	IBM Tivoli Directory Server for z/OS Server Plug-in Sample	163
7.6.1	Stepping through plugin_sample.c	163
7.6.2	Steps for building and running the sample plug-in	165
7.7	Exploiters of IBM Tivoli Directory Server for z/OS Plug-in Support	166
Chapter 8. Workload Management		167
8.1	Workload Management Overview	168
8.2	Using Configuration Options	168
8.2.1	Configuring WLM to support incoming requests	169

8.2.2	Configuring LDAP to exploit WLM.	170
8.3	Using Workload Manager and Operations Monitor together.	171
8.4	Workload Manager Health	171
Part 3.	Installation and configuration examples	173
Chapter 9.	Implementing IBM Tivoli Directory Server on a single system	175
9.1	A basic IBM Tivoli Directory Server server with LDBM	176
9.1.1	Prepare the z/OS system	176
9.1.2	Implementing IBM Tivoli Directory Server with dsconfig.	176
9.1.3	Starting and verifying IBM Tivoli Directory Server operation.	178
9.2	A basic IBM Tivoli Directory Server server with TDBM	178
9.2.1	Prepare the z/OS system	179
9.2.2	DB2 setup for IBM Tivoli Directory Server	179
9.2.3	Implementing IBM Tivoli Directory Server with dsconfig.	181
9.2.4	Starting and verifying IBM Tivoli Directory Server operation.	184
9.3	Set up file-based GDBM to track changes	185
9.4	Set up DB2-based GDBM to track changes	187
9.5	A basic IBM Tivoli Directory Server server with SDBM.	190
9.6	Loading the IBM-supplied schema	192
9.7	Loading the IBM-supplied sample.ldif file	193
9.8	Securing the IBM Tivoli Directory Server administration ID	194
9.9	Using CRAM-MD5 and DIGEST-MD5 binds.	196
9.10	Enabling SSL authentication.	198
9.11	Password policy implementation.	210
Chapter 10.	Using IBM Tivoli Directory Server in a Parallel Sysplex	217
10.1	Setting up the LDBM back end for sysplex	218
10.1.1	Changes to the configuration file	218
10.1.2	Starting and verifying operation	218
10.2	Setting up the TDBM server for sysplex	220
10.2.1	Changes to the configuration file	220
10.2.2	Starting and verifying operation	221
10.3	Other shared back ends	223
10.4	Setup a shared GDBM to track changes.	223
10.5	Set up a shared CDBM for advanced replication and password policy.	225
Chapter 11.	Replication	229
11.1	Basic Replication.	230
11.1.1	Master - replica topology.	230
11.1.2	Peer to peer topology	234
11.2	Advanced Replication	240
11.2.1	Major replication topologies	240
11.2.2	Configuring replication topologies.	241
11.2.3	Master-Replica replication configuration in advanced replication.	245
11.2.4	Peer to peer replication topology configuration in advanced replication.	248
Chapter 12.	Using LDAP and HCD.	255
12.1	Hardware Configuration Definition (HCD) and LDAP	256
12.2	Securing IBM Tivoli Directory Server for z/OS HCD	258
12.3	Configuring HCD and LDAP	258
12.3.1	Setting up the IBM Tivoli Directory Server for z/OS	259
12.3.2	Setting up the HCD LDAP plug-in.	261
12.3.3	Integrating the LDAP schema for HCD.	264

12.4 Using HCD and LDAP	264
12.4.1 Authentication	264
12.4.2 Usage examples	265
Chapter 13. Monitoring	267
13.1 Server monitoring	268
13.1.1 Monitor search with scope=sub	268
13.2 Monitoring and managing advanced replication	276
13.2.1 Showing advanced replication configuration information:	276
13.2.2 Extended operations related to advanced replication	279
13.2.3 Monitoring advanced replication status.	280
13.3 Using activity logging	282
13.4 Operations monitor	284
13.5 Audit logging	285
Chapter 14. Debugging	287
14.1 Overview	288
14.2 Debugging problems	288
14.2.1 Debugging configuration problems	288
14.2.2 Using server debug modes	288
14.2.3 Using CTRACE in-memory records	289
Part 4. Appendixes	291
Appendix A. Sample plug-in code	293
Source code for plugin_sample.c	294
Appendix B. Sample C code	305
Description of sample code	306
Related publications	313
IBM Redbooks	313
Other publications	313
Online resources	313
How to get Redbooks	314
Help from IBM	314
Index	315

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 about 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 illustrate 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.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

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

CICS®	Parallel Sysplex®	System z®
DB2®	RACF®	Tivoli®
IBM®	RDN®	z/OS®
IMS™	Redbooks®	z/VM®
MVS™	Redbooks (logo)  ®	z9®
OS/390®	RMF™	

The following terms are trademarks of other companies:

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

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries 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

This IBM® Redbooks® publication examines the IBM Tivoli® Directory Server for z/OS®. IBM Tivoli Directory Server is a powerful Lightweight Directory Access Protocol (LDAP) infrastructure that provides a foundation for deploying comprehensive identity management applications and advanced software architectures.

This publication provides an introduction to the IBM Tivoli Directory Server for z/OS that provides a brief summary of its features and an examination of the possible deployment topologies. It discusses planning a deployment of IBM Tivoli Directory Server for z/OS, which includes prerequisites, planning considerations, and data stores, and provides a brief overview of the configuration process. Additional chapters provide a detailed discussion of the IBM Tivoli Directory Server for z/OS architecture that examines the supported back ends, discusses in what scenarios they are best used, and provides usage examples for each back end. The discussion of schemas breaks down the schema and provides guidance on extending it. A broad discussion of authentication, authorization, and security examines the various access protections, bind mechanisms, and transport security available with IBM Tivoli Directory Server for z/OS. This chapter also provides an examination of the new Password Policy feature. Basic and advanced replication topologies are also covered. A discussion on plug-ins provides details on the various types of plug-ins, the plug-in architecture, and creating a plug-in, and provides an example plug-in. Integration of IBM Tivoli Directory Server for z/OS into the IBM Workload Manager environment is also covered.

This publication also provides detailed information about the configuration of IBM Tivoli Directory Server for z/OS. It discusses deploying IBM Tivoli Directory Server for z/OS on a single system, with examples of configuring the available back ends. Configuration examples are also provided for deploying the server in a Sysplex, and for both basic and advanced replication topologies. Finally it provides guidance on monitoring and debugging IBM Tivoli Directory Server for z/OS.

The team who wrote this book

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

Karan Singh is a Project Leader at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM ITSO classes worldwide on z/OS.

Corey C Bryant is an Advisory Software Developer for IBM Tivoli Directory Server for z/OS. He holds a BS in Computer Science from Binghamton University and an MS in Information Technology from RPI.

Jonathan Cottrell, CISSP, is an Advisory Software Engineer who has worked for the past ten years on the IBM Tivoli Directory Server for z/OS Development and Level 3 teams. He joined IBM in 1998 after graduating from Clarkson University with a Bachelor of Science degree in Computer Engineering.

Gillian Gainsford is an Identity and Access Management specialist who works at the University of Auckland in New Zealand. Before that, Gillian worked at IBM for 12 years

specializing in IT security. She is a board member of the Auckland Chapter of ISACA, and holds CISSP and CISA designations.

Saheem Granados is an Advisory Software Engineer at the IBM Poughkeepsie Development Lab. He currently is a member of the ITDS for z/OS development team. His expertise includes Encryption and Java Security. Saheem holds a Master's Degree from Binghamton University and has been with IBM for 11 years. He is also a Certified Information Systems Security Professional (CISSP).

Robert Green is a Field Technical Sales Specialist with IBM. Robert has worked in the information systems field since 1990, beginning as a MVS™ applications programmer and analyst, and then a systems programmer with an information systems bureau supporting the insurance industry. He joined IBM in 1996, assigned to the OS/390® development lab in Poughkeepsie, NY, where he spent 10 years as a software engineer. His current assignment at IBM as an FTSS is to work with customers to use the mainframe for deploying applications that go beyond the traditional CICS®, DB2® and IMS™ applications. Robert leads customers to identify which applications are suitable candidates for the mainframe, and then assists with deploying those applications to take advantage of the strengths of the mainframe. Robert has a Master's degree in Information Science from the University of South Alabama.

Diane Lia is a staff software engineer who has worked on LDAP on z/OS for 10 years, spending five of those years as a developer and five years as a functional verification tester.

Nilesh T Patel is a Software Engineer in the IBM Software Group in India. He is a solution Advisor - Tivoli Security and Compliance Management Solutions, and has been working as a technical leader of the Tivoli Directory Server team. His areas of expertise include IBM Tivoli Directory Server, Tivoli Access Manager, Tivoli Identity Manager, Tivoli Directory integrator, and Virtual Member Manager. He has published technical papers within the IBM developer domain and integration modules on IBM Open Process Automation Library for Tivoli Security products. He has delivered many technical web-casts to educate customers on new features and integration of Tivoli Security products. Nilesh holds a degree in Information technology from Pune Institute of Computer Technology in Pune, India.

John M Walsh is a software engineer who has worked for the past eight years on IBM Tivoli Directory Server for z/OS Development and Level 3 teams. He joined IBM in 2002 after graduating from Binghamton University with a Bachelor of Science degree in Computer Science. His areas of expertise include networking, WLM, FFDC, and LDAP.

Thanks to the following people for their contributions to this project:

Robert Haimowitz, Richard Conway
International Technical Support Organization, Poughkeepsie Center

John C Jones, Deborah Mian, Jeff Smith
IBM

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

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 books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

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

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



Part 1

Overview

In this part we introduce the IBM Tivoli Directory Server for z/OS and provide planning information.



Tivoli Directory Server for z/OS

This chapter introduces IBM Tivoli Directory Server for z/OS, a powerful and authoritative enterprise directory infrastructure that is a critical enabler for enterprise security. It provides robust and advanced LDAP services using TCP/IP, and the directory is based on a client/server model that can be exploited from any LDAP-compliant client applications or middleware. Directory data can be stored in either z/OS UNIX files or in DB2 tables for a highly scalable implementation.

This chapter will discuss the basic structure of LDAP directory content and will also touch on the architecture of IBM Tivoli Directory Server for z/OS server deployments ranging from basic single-server to more complex parallel sysplex and advanced replication architectures. The chapter will provide a basic understanding and reference point for these architectures before getting into the configuration details.

1.1 z/OS LDAP - features

The z/OS Lightweight Directory Access Protocol server is a part of the IBM Tivoli Directory Server for z/OS that can provide LDAP services for z/OS and non-z/OS hosted applications. It supports role based and fine-grained access control, and allows for delegated ownership of entries. Its features include:

- ▶ The dsconfig utility and CDBM back end to make the configuration process easier.
- ▶ Multiple concurrent database instances known as back ends that allow a single z/OS LDAP server to respond to requests from many logically separate portions of the LDAP tree.
- ▶ Hosting to any security data made available by TCP/IP and the LDAP directory, providing a network-accessible data repository accessible by other LDAP implementations.
- ▶ The optional use of RACF® to provide remote security services using LDAP protocols to software components that are not usually compatible or interoperable with RACF.
- ▶ Ability to securely encrypt the values stored within the directory using the latest encryption algorithms to protect directory data
- ▶ Encryption of data to and from LDAP clients using the z/OS Cryptographic Services System SSL. The LDAP server supports the Start TLS extended operation to switch a non-secure connection to a secure connection.
- ▶ Support for PKCS#11 hardware key storage and cryptographic acceleration
- ▶ Simple, CRAM-MD5, DIGEST-MD5, Kerberos (GSSAPI), and SASL EXTERNAL authentication
- ▶ Auditing capabilities through the GDBM change log back end and optionally the ICTX plug-in to cut SMF records
- ▶ Robust replication to enhance performance and offer flexible deployment options
- ▶ Filtered ACLs to provide greater flexibility in setting up authorisation within IBM Tivoli Directory Server.

1.2 IBM Tivoli Directory Server for z/OS

A directory is used to organize and store data that is expected to be read much more frequently than it is updated. A common example of a directory is a phone book, where information such as names and phone numbers are stored for residents of a specific location. Without a phone book, this information would likely be unorganized and difficult to access.

LDAP is based on the distributed client/server model and defines a standard method for accessing and updating directory information. The information is stored in a repository that is accessed and updated by the server based on client requests. Client requests are typically initiated from a user or application, such as an employee searching an employee directory, or an ATM's software verifying a PIN number for a transaction.

The IBM Tivoli Directory Server for z/OS deliverable that ships with the base of z/OS provides a Version 3 LDAP client and server. The z/OS LDAP client contains C APIs and command line utilities used to add, delete, modify, rename, compare, and search entries in an LDAP directory. The z/OS LDAP server is used to manage directory entries.

IBM Tivoli Directory Server for z/OS LDAP directory servers can range in complexity from small single-server deployments to larger and more complex multi-server deployments where data can be replicated among multiple servers and networks for increased availability, reliability, and performance.

1.3 Directory Architecture

An LDAP directory consists of a set of objects, otherwise known as entries, arranged in a hierarchical fashion to form a directory information tree (DIT). The entries are arranged such that high-level entries represent the entries that fall below them. An entry typically describes a person, place, or thing, and consists of a collection of attributes. Each attribute has a type and one or more values. In the case of a person, the attributes that describe the person could be the person's common name (using the **cn** attribute type) and telephone number (using the **telephoneNumber** attribute type).

Each entry has a name that is called a distinguished name (DN), which is unique among all other entries in the directory. A DN is formed by taking the name of the entry itself, called the relative distinguished name (RDN@) and concatenating the names of its ancestor entries. For example, the entry for Barb Lee in the diagram in Figure 1-1 has an RDN of cn=Barb Lee and a DN of cn=Barb Lee, o=IBM, c=US. The unique DN allows users to refer to a specific entry using its DN.

Figure 1-1 shows the structure of entries and attributes within a sample LDAP DIT.

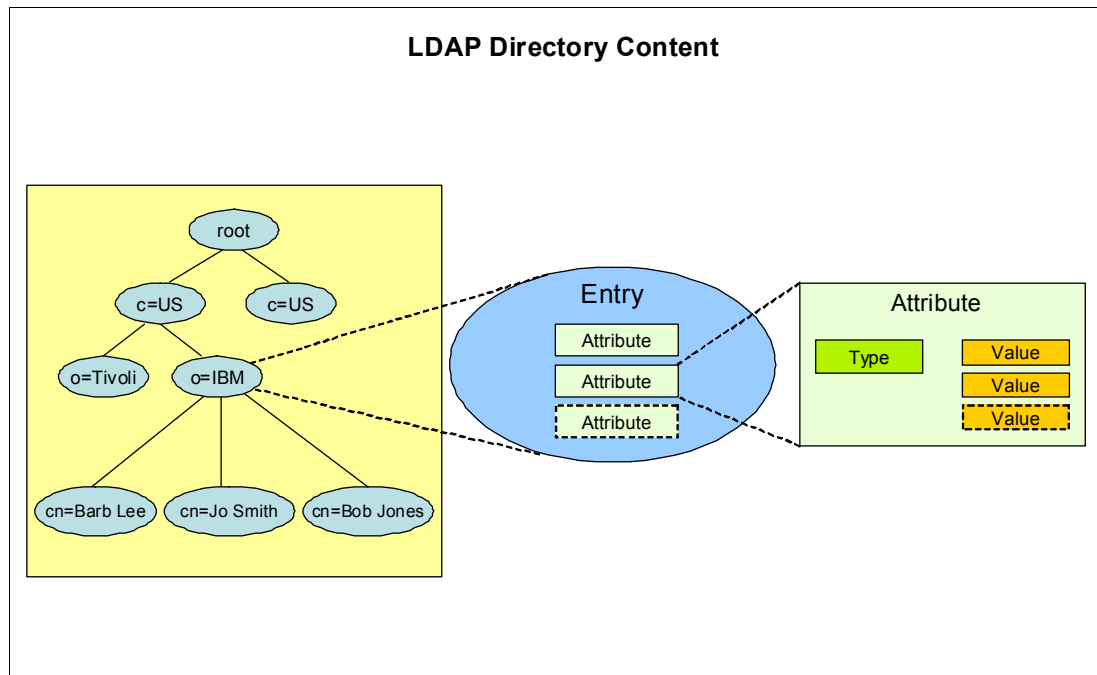


Figure 1-1 Sample LDAP directory information tree

1.4 Server Architectures

The IBM Tivoli Directory Server for z/OS LDAP server can be configured to run in one of a number of operational modes:

- ▶ Single-server mode: A single LDAP server running on a system.
- ▶ Multiple single-server mode: Multiple LDAP servers running individually on the same system.
- ▶ Multi-server mode: Multiple LDAP servers running together in a parallel sysplex.

In all of these operational modes, the server can perform basic or advanced replication. However, each server in multiple single-server operational mode must have its own separate set of replica servers, whereas each server in multi-server operational mode must have the same set of replica servers.

Advanced replication allows multiple LDAP servers to synchronize their data. Specific subtrees within the directory can be selected to participate in an advanced replication topology. These subtrees can be designated with roles such as a supplier or a consumer. Replication topology choices can be combined to serve many directory architectures, allowing flexibility in architecting solutions that enable data redundancy, server availability, and scalability requirements. The following four advanced replication topologies can be deployed with IBM Tivoli Directory Server for z/OS LDAP servers:

- ▶ Master-replica replication: Provides a read-only backup of replicated subtrees and reduces search workload.
- ▶ Forwarding replication: Relieves replication workload from master servers in a network containing many widely-distributed replicas and reduces search workload.
- ▶ Peer-to-peer replication: Provides a local server for handling updates in a widely distributed topology, and also provides a backup master server that can take over if necessary.
- ▶ Gateway replication: Primarily used to reduce network traffic for a widely distributed topology.

1.4.1 Single-Server

The single-server IBM Tivoli Directory Server for z/OS architecture forms the foundation for all other architectures that can be deployed with IBM Tivoli Directory Server for z/OS.

The LDAP server can be configured to use one or more of a number of back end data stores:

- ▶ SDBM: The SDBM back end provides remote LDAP access to user, group, connection, and general resource profile information stored in RACF.
- ▶ LDBM: The LDBM back end provides a file-based back end to store directory information in a UNIX System Services file system. LDBM is a general-purpose back end that can store any type of directory information.
- ▶ Schema: The directory schema back end contains a set of rules and constraints concerning directory information tree structure, object class definitions, attribute types, and syntaxes that characterize the directory information base (DIB).
- ▶ CDBM: The CDBM back end stores configuration information, such as for advanced replication and password policy. CDBM is file-based, storing its directory information in a UNIX System Services file system.
- ▶ GDBM: The GDBM back end manages change log entries created as a result of changes to the LDAP schema or to entries in other back ends, including RACF entries. The change log entries can be kept in UNIX System Services files or in a DB2 database.
- ▶ TDBM: The TDBM back end provides a DB2-based back end to store directory information. TDBM is a general-purpose back end that can store any type of directory information.
- ▶ Plug-in: A plug-in is a software module that augments the functionality of the IBM Tivoli Directory Server for z/OS LDAP server. For example, a client operation plug-in can be written to handle specific client operation requests rather than allowing a configured database back end to handle the requests.

In a single-server operational mode, only a single instance of the LDAP server can use a given TDBM, LDBM, CDBM, or GDBM database to store directory data.

In multiple single-server operational mode, two or more LDAP servers can run in single-server mode on the same system, each with separate TDBM, LDBM, CDBM, or GDBM back ends.

Figure 1-2 shows an LDAP client connecting to a IBM Tivoli Directory Server for z/OS LDAP server running in single-server mode with all of the aforementioned back ends configured.

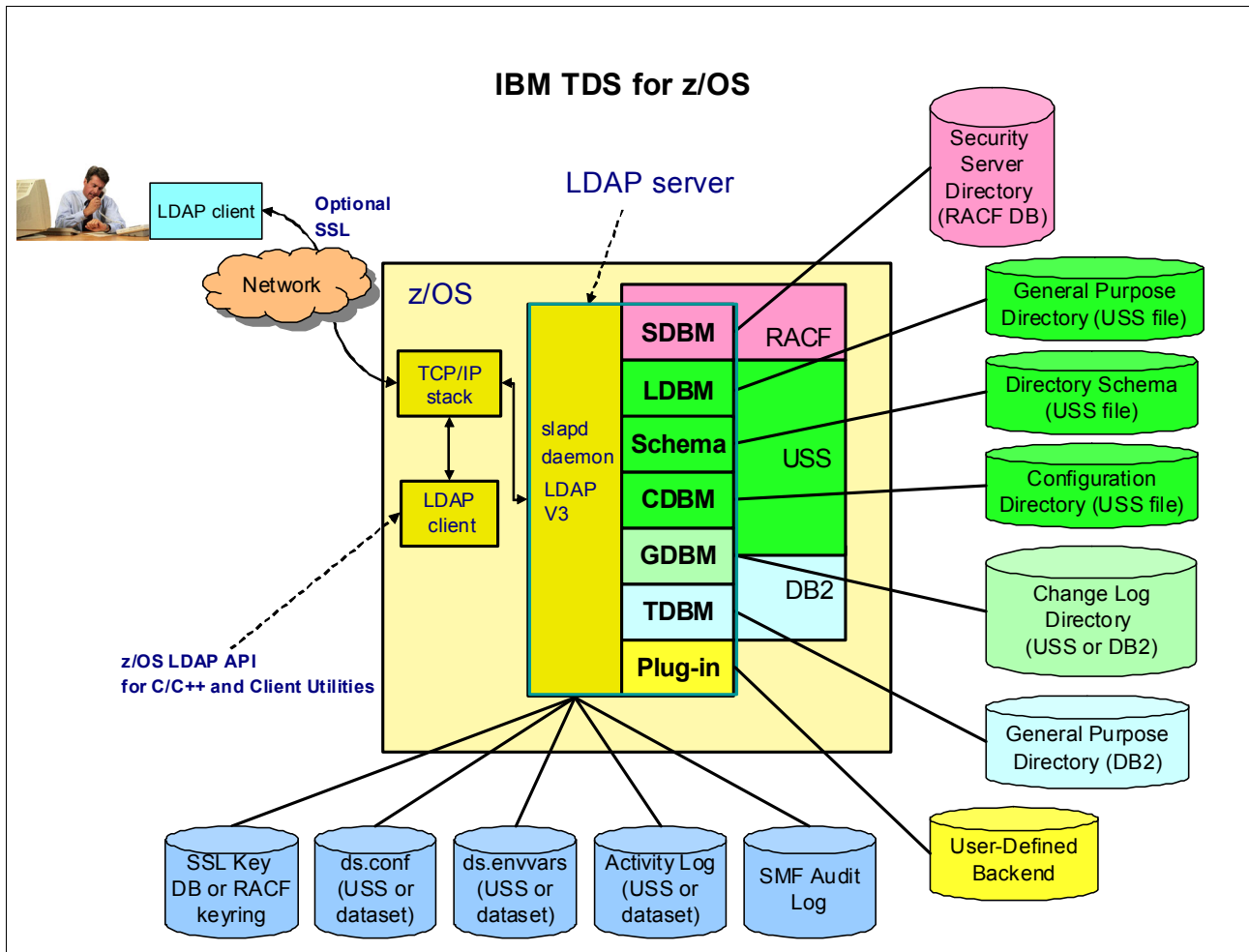


Figure 1-2 Single-server IBM Tivoli Directory Server with back ends configured

The LDAP server configuration file (ds.conf) contains configuration options that are read once when the LDAP server is started. There are also a number of environment variables that are processed by the LDAP server and utilities. The majority of these can be specified in the LDAP server environment variables file (ds.envvars).

The LDAP server runs as a daemon. In other words, when the server is started, the process runs unattended waiting for client requests to come in, and performs services based on those client requests.

The LDAP client connects to the LDAP server over TCP/IP, using an LDAP API and requests an operation. The most frequently used LDAP operation is a search of the directory. The server then attempts to perform the operation and responds with the results of the request

processing, or with a reference to another LDAP server where the application can get the data it has requested.

LDAP client requests can be performed using an anonymous identity or the LDAP bind operation can be used to supply an authentication identity. This authentication process can be used by distributed applications that need to implement authentication. The LDAP server can then use the identity to perform authorization checking when accessing entries in the directory.

An Access Control List (ACL) protects information stored in an LDAP directory. An ACL is used to restrict access to portions of the directory, to specific directory entries, or to specific attributes within an entry. Access control can be specified for individual users, groups, and even specific bind or entry access information such as the client's IP address or entry access time of day.

LDAP has the ability to protect LDAP access with Secure Sockets Layer (SSL) and Transport Layer Security (TLS), which use public-key infrastructure (PKI) algorithms to establish and maintain encrypted communication between a client and server. In order for the LDAP client to communicate with an LDAP server over an SSL/TLS-protected TCP/IP socket connection, the LDAP server must transmit a certificate to the LDAP client and, optionally, the client can transmit its certificate to the LDAP server.

The LDAP client and server verify the certificates sent to them by using public-key digital signatures, by which they take a certificate and compare the digital signature in the certificate with a signature they compute based on having the public-key of the signer of the certificate. To do this, the LDAP client and server must have the public-key of the signer of the certificate, which is stored in a key database, RACF key ring, or a PKCS #11 token. The same repositories are used to store the certificates that the client or server will transmit to each other during the startup of the SSL/TLS-protected communications.

The LDAP server can log client activity in an activity log file. This file can be referenced and analyzed to understand the client operations that have been handled by the server. The type of information contained in an activity log includes: operation type, client IP address, server messages, and activity summary statistics.

The LDAP server can generate SMF type-83 subtype 3 audit records. The audit records contain information provided on LDAP client operation requests, and can be configured to write audit records when an operation successfully completes, fails, or for both cases. These audit records can be unloaded using the RACF SMF data Unload utility for further analysis by auditing tools.

The functionality of the LDAP server can be augmented by user-written software plug-ins. When a client request is received by the LDAP server, the server initially attempts to call a configured database back end to process the request. If a matching database back end is not found, the LDAP server attempts to call a configured plug-in to process the request. Plug-ins can be designed to execute at one of three points of client request processing:

- ▶ Before a client request is processed
- ▶ To handle the actual request processing
- ▶ After a client request completes

1.4.2 Multi-Server (Sysplex)

In multi-server operational mode, multiple concurrent instances of the LDAP server use the same TDBM, LDBM, CDBM, or GDBM database to store directory data. The LDAP servers can run on the same host system, or on separate host systems. In both cases, the z/OS

systems are connected in a parallel sysplex, and use XCF (Cross-system coupling facility) messaging to facilitate communication between the LDAP servers.

Multi-server mode is intended for use in an environment where high transactional volume is common, or where maximum availability is required. This mode provides benefits of improved reliability, availability, performance, and resource utilization. These benefits are achieved by enabling concurrent running of multiple servers that are functionally equivalent and that provide access to the same LDAP directory data.

The LDAP server provides Workload Manager (WLM) support, allowing an installation to set performance goals for work within the LDAP server. The work load manager will perform load balancing by routing requests to each of the servers in a sysplex group so that work is shared across each of the servers in the sysplex. If more capacity is required, another server can be brought up. If one of the LDAP servers in the sysplex goes down, WLM will route requests to the remaining servers. The servers can also be configured to automatically restart on failure (ARM support).

In Figure 1-3, a client connects to a IBM Tivoli Directory Server for z/OS server running in multi-server mode.

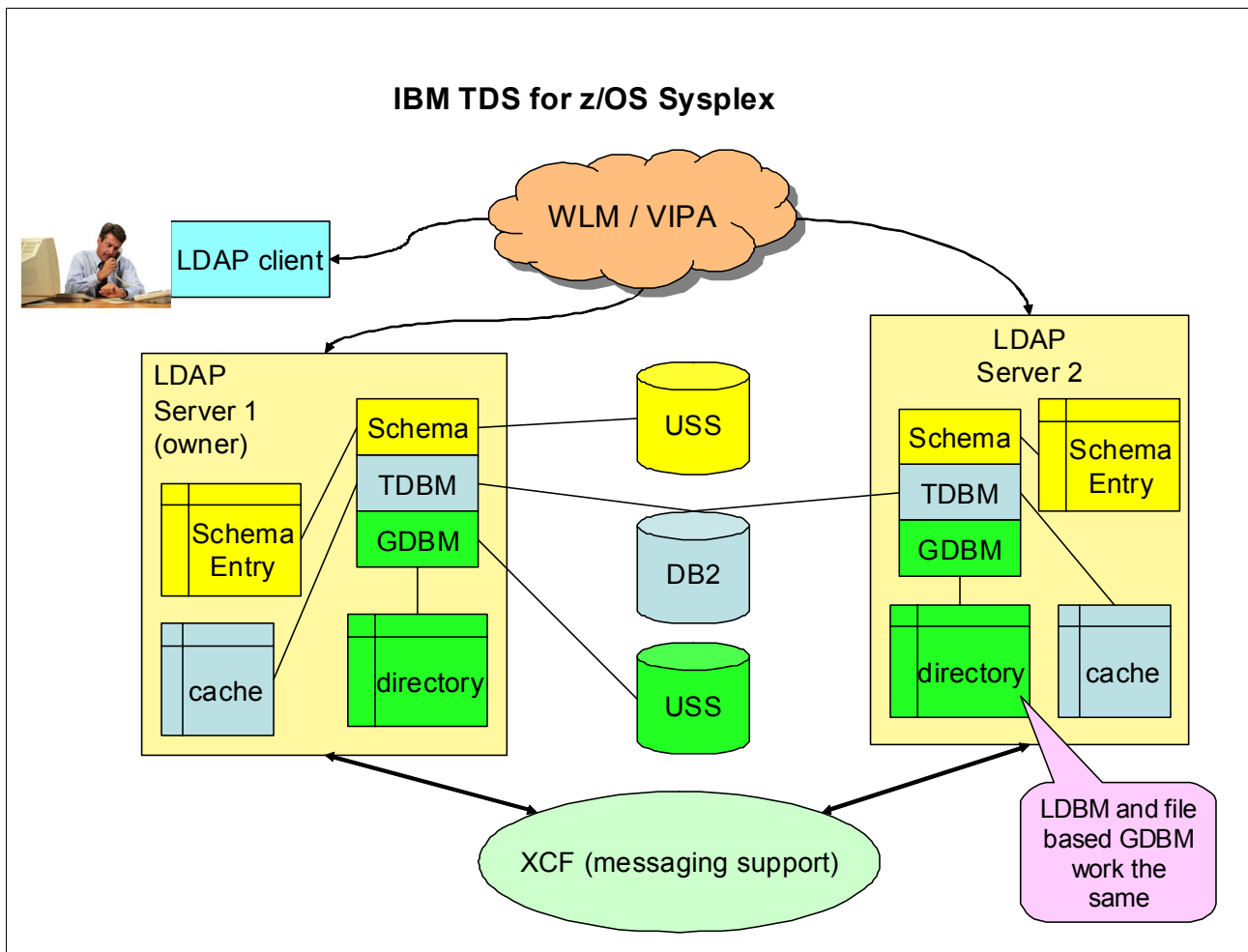


Figure 1-3 Multi-server mode

Updates to one server are communicated to all other servers through the cross-system coupling facility (XCF). This enables all servers to know about the updates.

In multi-server mode, each LDAP server will keep its own copy of the shared TDBM back end in caches, and each server writes to the TDBM back end because of DB2 data sharing. After a server updates the TDBM back end, it will notify all other servers of the change using XCF. The behavior of DB2-based GDBM is similar to that of TDBM.

UNIX System Services based back end files (schema, LDBM, CDBM, and file-based GDBM) can also be shared among servers in multi-server mode. However, only the sysplex owner can write to these back ends. Non-sysplex owner servers maintain in-memory copies of the UNIX System Services based files and directories. If changes are directed to a non-sysplex owner, that server uses XCF to forward the change to the sysplex owner. The owner then makes the change in memory and in the back end, and broadcasts the change to the other servers using XCF. The other servers then update their directory in memory. Note that all servers must point to, and have access to, the same back end location in a shared file system. If the sysplex owner goes offline, another LDAP server in the sysplex group will become the owner.

1.4.3 Master-Replica Replication

The most basic advanced replication topology is a master-server replication (otherwise known as supplier-consumer). The master is a writable server, meaning that it is able to receive updates from clients. One or more subtrees within the master server are designated to participate in replication. The replicated data could be the entire directory or just a small subtree of the directory. The read-only replica server contains a copy of the subtree that is replicated from the master server. The master-replica topology enables data redundancy by providing read-only backups of selected subtrees, and can reduce search workloads that can now be targeted at any server in the topology.

A master server can have several replicas, with each replica containing either a copy of the master's entire directory, or just a subtree of the directory. A replication context identifies the portion of the DIT that is to be replicated from one server to another.

Figure 1-4 shows a sample Master-Replica topology.

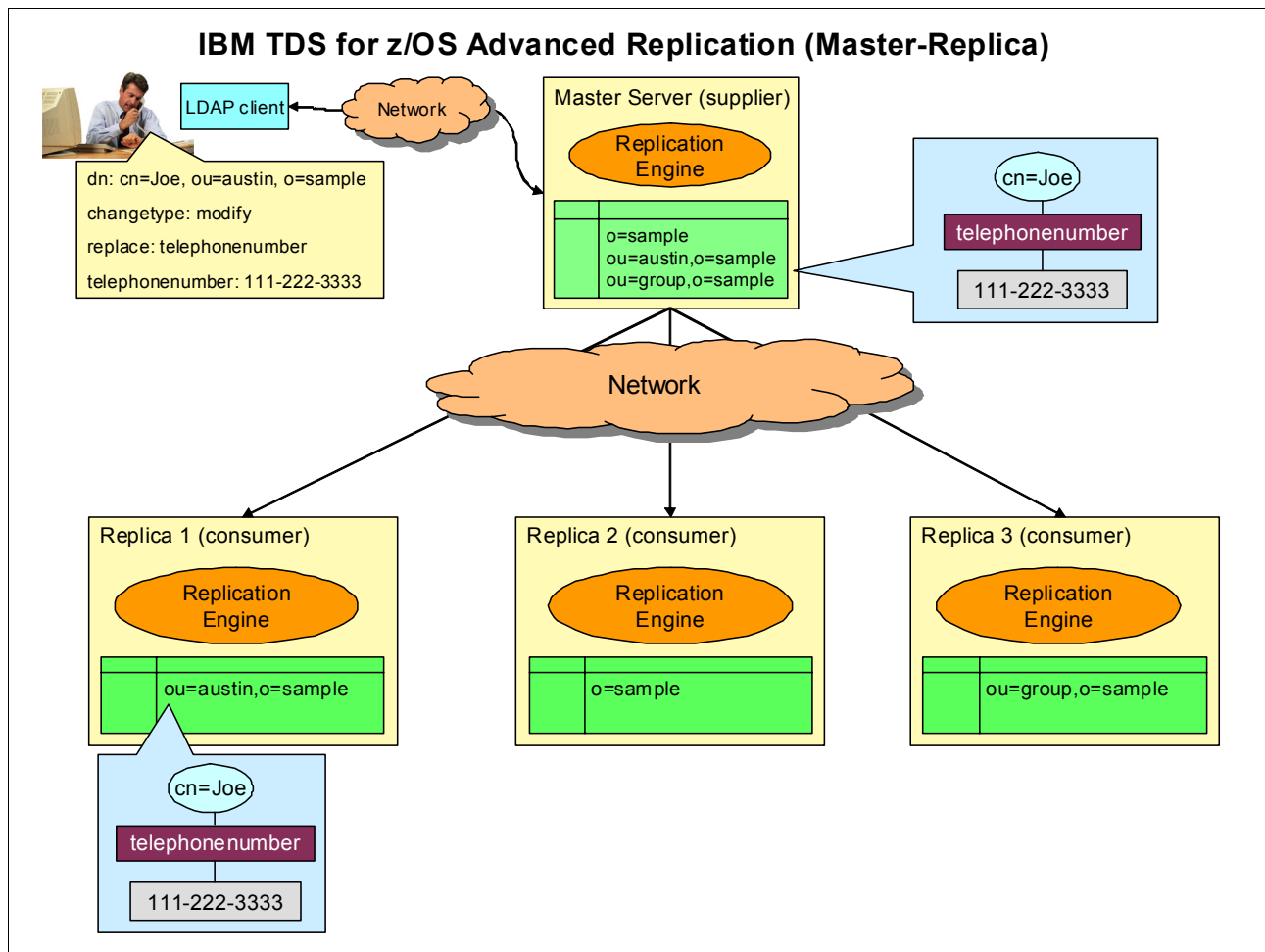


Figure 1-4 Master-Replica topology

In the diagram shown in Figure 1-4, the following three replication contexts are in effect:

- ▶ o=sample
- ▶ ou=austin,o=sample
- ▶ ou=group,o=sample

The server labeled Replica 2 contains a copy of the entire Master Server directory. The servers labeled Replica 1 and Replica 3 each contain a copy of separate subtrees from the Master Server's directory.

The flow begins as the client requests a modify of the telephonenumber attribute for cn=Joe, ou=austin, o=sample. The update is directed to the Master Server because it is the only writable server in the topology. If the update were directed to one of the read-only Replica servers, they would refer the client to the Master Server. The Master Server queues the update along with any other pending updates within its replication engine.

When replication is designated to occur for the ou=austin, o=sample subtree, all queued modifications, including that for cn=Joe, ou=austin, o=sample, will then be replicated to Replica 1. Note that the replication of subtree ou=austin, o=sample does not cause replication to the Replica 2 server. The update for cn=Joe, ou=austin, o=sample, would be replicated to Replica 2 when replication is designated to occur for the o=sample subtree.

1.4.4 Forwarding (Cascading) Replication

Forwarding (cascading) replication is an advanced replication topology that consists of multiple tiers of LDAP servers. A master server replicates to a set of read-only forwarding servers that in turn replicate to other servers. A forwarding topology enables off-loading of replication work from the master server, for example in a network containing many widely distributed replicas. A forwarding topology can also enable reduced search workload because searches can be targeted at any server in the topology.

Figure 1-5 shows a sample Forwarding topology.

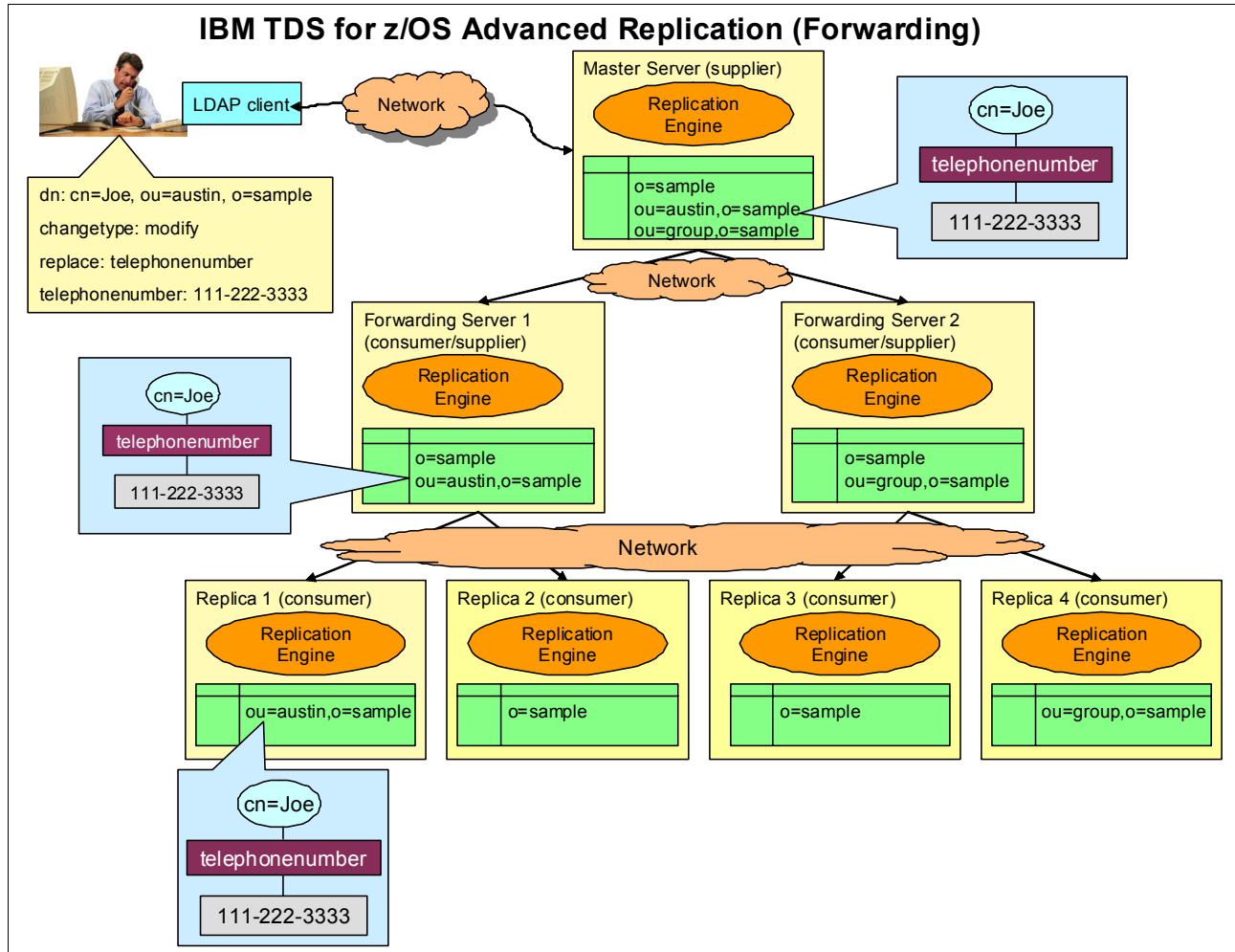


Figure 1-5 Forwarding topology

In this case, the master server is a supplier to the two forwarding servers. The forwarding servers serve as consumers of the master server and suppliers to the replica servers that are associated with them. The replica servers serve as consumers of their respective forwarding servers.

The following three replication contexts are in effect for the server labeled Master Server:

- ▶ `o=sample`
- ▶ `ou=austin,o=sample`
- ▶ `ou=group,o=sample`

The following two replication contexts are in effect for the server labeled Forwarding Server 1:

- ▶ o=sample
- ▶ ou=austin,o=sample

The following two replication contexts are in effect for the server labeled Forwarding Server 2:

- ▶ o=sample
- ▶ ou=group,o=sample

The flow begins as the client requests a modification of the telephonenumber attribute for cn=Joe, ou=austin, o=sample. The update is directed to the Master Server because it is the only writable server in the topology. If the update were directed to one of the forwarding or replica servers, they would refer the client to the Master Server. The Master Server queues the update along with any other pending updates within its replication engine.

When replication is designated to occur for the ou=austin, o=sample subtree of the Master Server, all queued modifications, including that for cn=Joe, ou=austin, o=sample, will then be replicated to Forwarding Server 1. Forwarding Server 1 then queues the updates along with other pending updates within its replication engine.

When replication is designated to occur for the ou=austin, o=sample subtree of Forwarding Server 1, all queued modifications, including that for cn=Joe, ou=austin, o=sample, will then be replicated to Replica Server 1.

1.4.5 Peer-to-Peer Replication

Peer-to-peer replication allows for several servers to act as master servers, with each master responsible for updating other master servers and replica servers. Peer-to-peer replication allows for performance improvements by providing a local server to handle updates in a widely distributed network. Availability and reliability are also improved by providing a backup master server that can take over immediately if the primary master fails. This topology can also help reduce the search workload as clients can send search requests to all the servers.

Figure 1-6 shows a sample Peer-to-Peer topology.

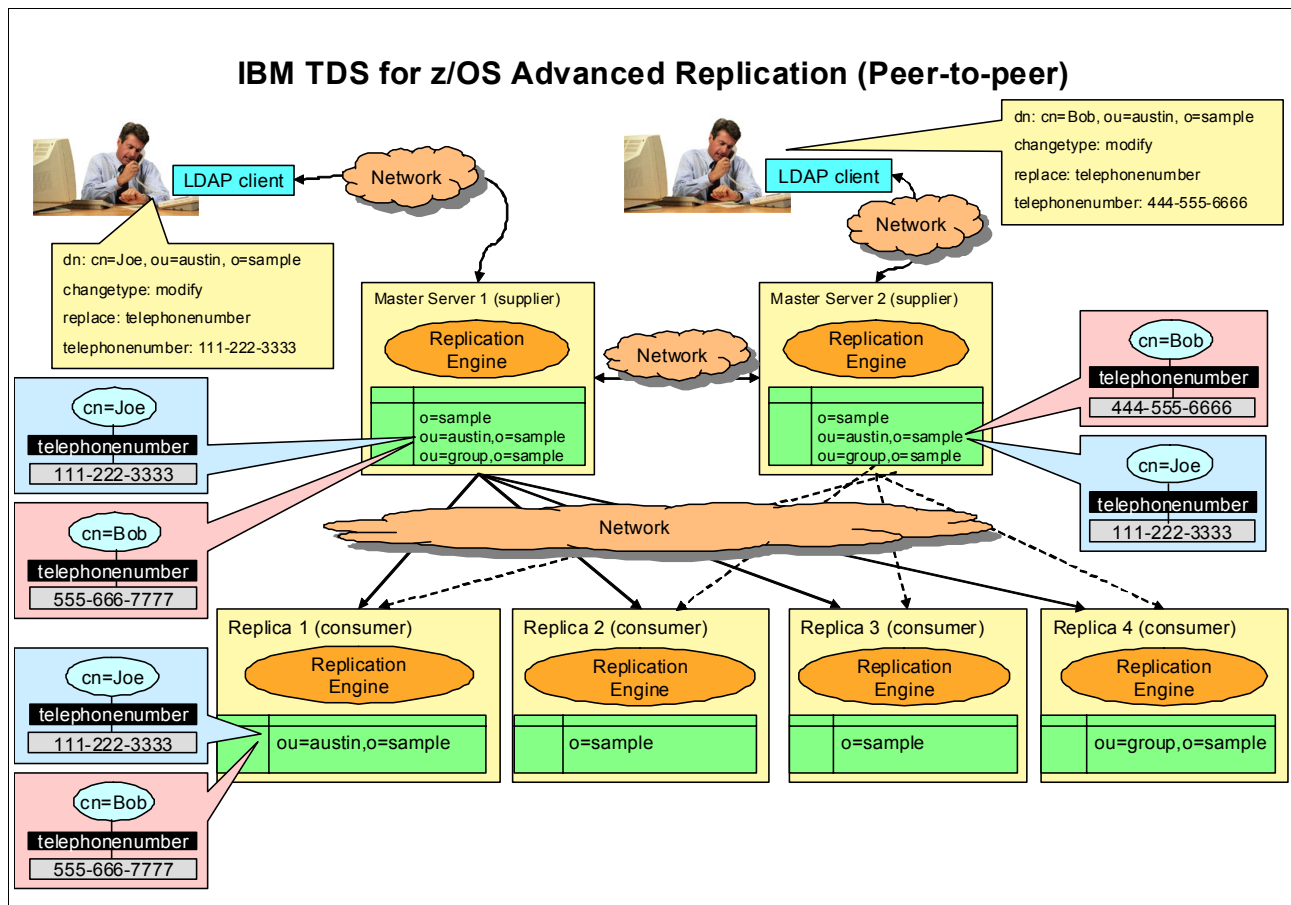


Figure 1-6 Peer-to-peer topology

The servers labeled Master Server 1 and Master Server 2 are the peer master servers. Peer master servers only replicate client changes that were originally requested of them. They do not replicate updates received from other peer master servers. Peer master servers must connect to all other consumers in the topology to ensure the consumers receive all client updates.

The following three replication contexts are in effect for the servers labeled Master Server 1 and Master Server 2:

- ▶ o=sample
- ▶ ou=austin,o=sample
- ▶ ou=group,o=sample

The flow begins as one client requests a modify of the telephonenumber attribute for cn=Joe, ou=austin, o=sample on Master Server 1. Master Server 1 queues the update along with any other pending updates within its replication engine.

Another client requests a modify of the telephonenumber attribute for cn=Bob, ou=austin, o=sample on Master Server 2. Master Server 2 queues the update along with any other pending updates within its replication engine.

When replication is designated to occur for the ou=austin, o=sample subtree on Master Server 2, all queued modifications, including that for cn=Bob, ou=austin, o=sample, will then be replicated to both Master Server 1 and Replica 1.

When replication is designated to occur for the ou=austin, o=sample subtree on Master Server 1, all queued modifications, including that for cn=Joe, ou=austin, o=sample, will then be replicated to both Master Server 2 and Replica 1.

1.4.6 Gateway Replication

Gateway replication uses gateway servers to collect and distribute replication information across the replication sites of a replicating network. The primary use of a gateway replication is to reduce network traffic. A replication site consists of a gateway server and any master, peer, or replica servers configured to replicate together.

A gateway server is a writable master server that acts as a peer server within its replication site. In other words, it can receive and replicate client updates and updates from the other peer-master servers within its replication site.

Within the gateway network, the gateway server acts as a two-way forwarding server. In one direction, the peers in its replication site act as the suppliers to the gateway server and the other gateway servers are its consumers. In the other direction, other gateway servers act as suppliers to the gateway server and the servers within its own replication site act as consumers. Note that when a gateway server receives updates from another gateway server, it will only replicate those updates to servers within its own replication site.

Figure 1-7 shows a sample gateway topology consisting of four replication sites.

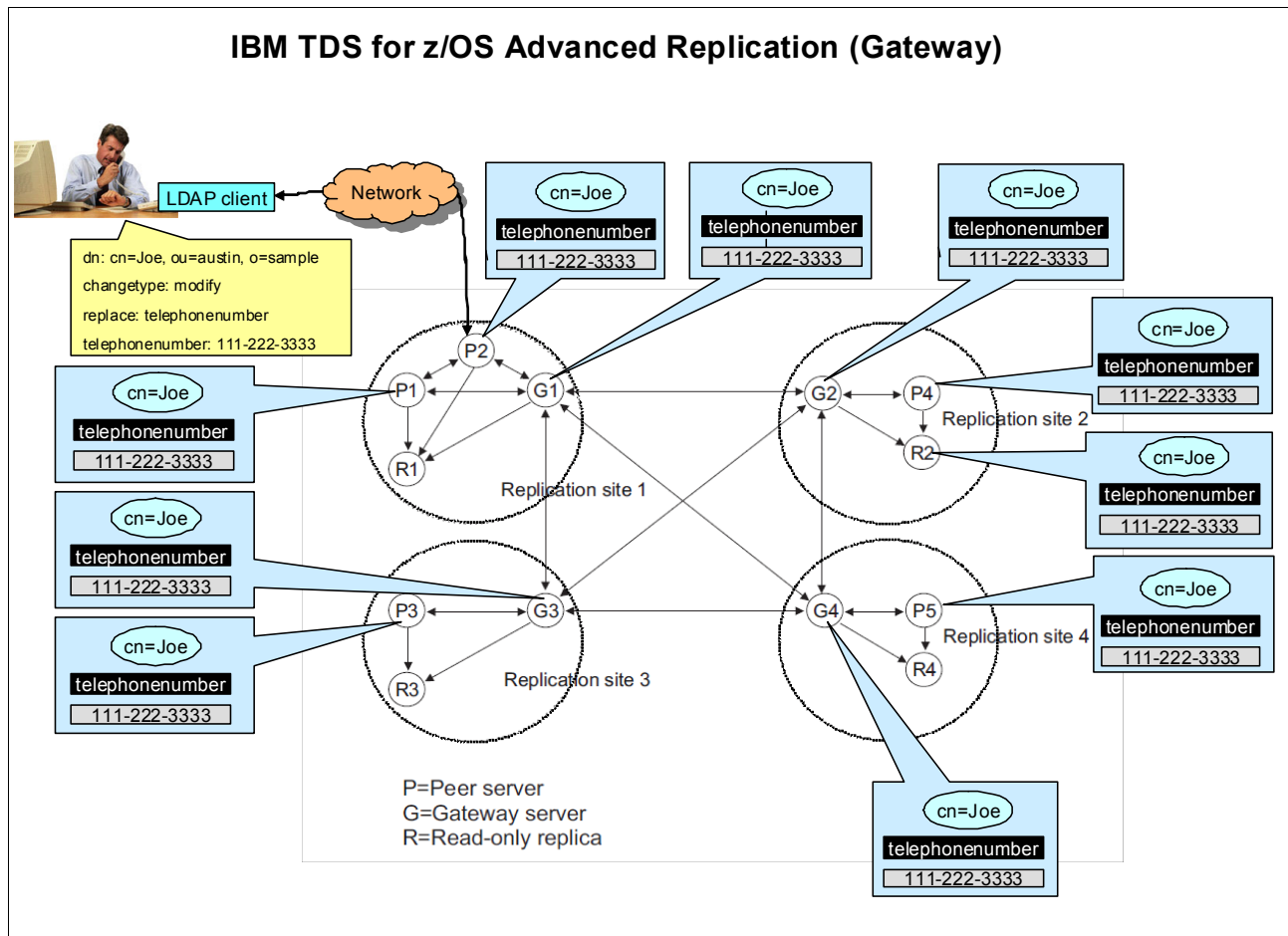


Figure 1-7 Gateway topology

The replication contexts that are in effect for the Gateway Servers labeled G1-G4 and the Peer Servers labeled P1-P5 are as follows:

- ▶ o=sample
- ▶ ou=austin,o=sample
- ▶ ou=poughkeepise,o=sample
- ▶ ou=raleigh,o=sample
- ▶ ou=rochester,o=sample

The Replica Servers labeled R1-R4 contain the following subtrees:

- ▶ R1: ou=poughkeepise,o=sample
- ▶ R2: ou=austin,o=sample
- ▶ R3: ou=raleigh,o=sample
- ▶ R4: ou=rochester,o=sample

The flow begins when a client requests a modify of the telephonenumber attribute for cn=Joe, ou=austin, o=sample on Peer Server P2. Peer Server P2 queues the update along with any other pending updates within its replication engine.

When replication is designated to occur for the ou=austin, o=sample subtree on Peer Server P2, all queued modifications will be replicated to Peer Server P1 and Gateway Server G1. Gateway Server G1 queues the update along with any other pending updates within its replication engine.

When replication is designated to occur for the ou=austin, o=sample subtree on Gateway Server G1, all queued modifications will be replicated to Gateway Servers G2, G3, and G4. Gateway Server G2 queues the update along with any other pending updates within its replication engine.

When replication is designated to occur for the ou=austin, o=sample subtree on Gateway Server G2, all queued modifications will be replicated to Peer Server P4 and Replica Server R2. Note that Gateway Server G2 only replicates the change throughout its site, and does not replicate the change to other gateway servers.

When replication is designated to occur for the ou=austin, o=sample subtree on Gateway Servers G3 and G4, all queued modifications will be replicated to Peer Server P3 and Peer Server P5, respectively.

1.4.7 Remote security services using the z/OS LDAP server

RACF can provide remote security services to applications that are requested using the LDAP protocols. They require that IBM Tivoli Directory Server be active on the z/OS that hosts the RACF instance, and the SDBM back end and the ICTX plug-in to be configured. The following services are made available by RACF on z/OS to off-z/OS applications:

- ▶ Remote authorization
- ▶ Remote auditing
- ▶ Remote identity cache.

Note: The LDAP client must support extended operations (EXOP) and Distinguished encoding rules (DER) in order for the LDAP client to connect to a remote application



Planning

This chapter provides planning information for deploying IBM Tivoli Directory Server for z/OS.

2.1 Planning and Considerations

Before deploying IBM Tivoli Directory Server for z/OS in your IT system environment, there are things to consider before you configure it and start fully using it. By considering these things up front, it is less likely you will have to do extensive changes after the LDAP server is already in production. This chapter discusses what needs to be considered before configuring and deploying IBM Tivoli Directory Server for z/OS.

2.2 Where to store your data

IBM Tivoli Directory Server for z/OS has several backing stores or back ends where data can reside. The supported back ends are:

- ▶ LDBM is a file-based back end where the data resides in files stored in a z/OS UNIX System Services file system, such as zFS. This back end is a general purpose back end that allows user-customizable data (as long as it adheres to the LDAP server's current schema) to be stored in the directory. The LDBM back end is meant for no more than 250,000 entries while running in 31-bit mode, and no more than 500,000 entries while running in 64-bit mode. Multiple LDBM back ends are allowed to be configured in IBM Tivoli Directory Server for z/OS.
- ▶ TDBM is a DB2-based back end where the data resides in DB2 tables. Like the LDBM back end, the TDBM back end is a general purpose back end where any user customizable data is allowed to be stored in the directory. Multiple TDBM back ends can be configured.
- ▶ SDBM is a back end that interfaces directly with the RACF database and allows access to RACF users, groups, user-group connections, general resource profiles, and SETROPTS system settings affecting RACF general resource classes. This back end provides a RACF administrator the ability to remotely manage RACF data using the LDAP protocol.
- ▶ GDBM is a file-based or DB2-based back end that logs updates made to other back ends and in RACF. This back end is also commonly referred to as the changelog back end.
- ▶ CDBM is a specialized file-based back end that stores configuration-related entries for the advanced replication (z/OS V1R11) and password policy (z/OS V1R12) features in the LDAP server. See 1.4, “Server Architectures” on page 5 and 5.6, “Password Policy” on page 127 for more information about these features.
- ▶ EXOP (extended operations) is a special back end that is used to access data in z/OS Policy Director. The extended operations back end supports two extended operations, GetDnForUserid and GetDNForPrivileges, for use with z/OS Policy Director.
- ▶ Plug-ins extend the existing function of the LDAP server. There are three types of plug-ins supported in IBM Tivoli Directory Server for z/OS: pre-operation, post-operation, and client-operation.

See Chapter 3, “Back ends” on page 29 and Chapter 7, “Plug-ins” on page 155 for more detailed information about these back ends and plug-ins.

2.3 Required products

This section discusses the required products that are necessary for running the IBM Tivoli Directory Server for z/OS:

- ▶ **z/OS UNIX System Services file system:** A z/OS UNIX System Services file system is needed to store the schema back end. When IBM Tivoli Directory Server for z/OS is started for the first time, the initial or minimum schema is created in the directory specified by the **schemaPath** configuration option or in `/var/ldap/schema` if the **schemaPath** configuration option is not specified. Generally, change the default directory to one that uses a separately mounted file system. The minimum schema allows the SDBM, GDBM, and CDBM back ends to be used without any additional modifications.

See Chapter 4, “Schemas” on page 73 for more information about the schema.

- ▶ **WLM (Workload Manager):** WLM must be configured and installed to allow the IBM Tivoli Directory Server for z/OS on V1R11 or later to run. WLM allows performance goals to be set for work within the LDAP server. By default, all LDAP server work goes to the GENERAL transaction name, which might need to be further tuned in your environment. See Chapter 8, “Workload Management” on page 167 for more information about tuning and configuring WLM for use with the LDAP server.

2.3.1 Optional products

This section discusses the optional products that might need to be installed or configured on your system based on the features that you are planning on using in the IBM Tivoli Directory Server for z/OS.

- ▶ **TDBM or DB2-based GDBM back ends:** DB2 and ODBC must be installed and configured to use the TDBM and DB2-based GDBM back ends to be usable. See *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for more information about installing and configuring DB2 and ODBC.
- ▶ **SDBM back end:** RACF (Resource Access Control Facility) must be installed to allow the use of the SDBM back end. The RACF Subsystem function of RACF must be defined and activated to allow the LDAP server to communicate with RACF through the SDBM back end. See *z/OS V1R12.0 Security Server RACF System Programmer's Guide*, SA22-7861 for more information.
- ▶ **LDBM, GDBM (file-based), or CDBM back ends:** A z/OS UNIX System Services file system is needed for each configured LDBM, GDBM (file-based), and CDBM back end. The location of the file-based back ends is controlled by the **databaseDirectory** configuration option in the back end specific section of the configuration file. Generally, use a separately mounted file system when configuring a file-based back end.
- ▶ **Secure communications:** The z/OS Cryptographic Services System SSL must be installed to use secure communications between your LDAP client applications and IBM Tivoli Directory Server for z/OS. See *z/OS V1R12.0 System SSL Programming*, SC24-5901 for more information about installing System SSL.

See 5.8, “SSL/TLS” on page 133 for more information about configuring IBM Tivoli Directory Server for z/OS to use SAF key rings, SSL key database files, or PKCS#11 tokens to store your SSL certificates.

- ▶ **AES and DES encryption:** IBM Tivoli Directory Server for z/OS supports using ICSF (Integrated Cryptographic Security Facility) for the AES and DES encryption of sensitive attribute values, such as `userPassword`, `secretKey`, `replicaCredentials`, `ibm-replicaKeyPwd`, and `ibm-slapdMasterPw`, in the LDBM, TDBM, and CDBM back ends. The AES or DES keys used by the LDAP server can be stored in the ICSF CKDS (Cryptographic Key Data Set) with the KGUP program. See 5.7, “Encryption and Hashing” on page 132 for more information about using the ICSF KGUP program.
- ▶ **Kerberos authentication:** IBM Tivoli Directory Server for z/OS allows users to authenticate to the directory using Kerberos authentication with the z/OS Integrated Security Services Network Authentication Service product that provides the IBM implementation of Kerberos

V5. See *V1R12.0 Network Authentication Service Administration*, SC24-5926 for more information about installing and configuring Kerberos.

IBM Tivoli Directory Server for z/OS supports running in single-server or multi-server operating modes. If IBM Tivoli Directory Server for z/OS is configured to run in multi-server mode, parallel sysplex must be available and the schema back end must be shared among all LDAP servers in the sysplex. See 1.2, “IBM Tivoli Directory Server for z/OS” on page 4 for more information.

2.4 Configuring IBM Tivoli Directory Server for z/OS

IBM Tivoli Directory Server for z/OS provides the **dsconfig** utility to help simplify the configuration of your own LDAP server. Before running the **dsconfig** utility, there are several additional features that should be considered before configuring your LDAP server:

- ▶ Where and how to store user passwords
- ▶ Advanced replication
- ▶ Password policy
- ▶ Activity and audit logging

2.4.1 Where and how to store user passwords?

If you have decided to configure an LDBM or TDBM back end, you have two choices for your user's passwords:

1. The userPassword attribute value of the user entries in the back end.
2. If there are already users and passwords defined in the z/OS Security Manager (e.g. RACF), the LDBM, TDBM, or CDBM back end can use these existing users and passwords without introducing another password repository. This is called native authentication because the underlying z/OS Security Manager does the password verification for the specified user and password when authenticating to the directory.

If you have decided to use the existing users and passwords already defined in the z/OS Security Manager, make certain to fill out the native authentication section in the **dsconfig** profile files. See 5.3, “Native authentication” on page 104 for more information.

If the passwords are to be stored in the entry's **userPassword** attribute value, there are several encryption or hashing methods that are available to protect the passwords in the directory:

- ▶ AES (Advanced Encryption Standard)
- ▶ DES (Data Encryption Standard)
- ▶ SHA (SHA-1)
- ▶ SSHA (Salted SHA-1 – only supported in z/OS V1R12 and later)
- ▶ crypt
- ▶ md5

SHA, SSHA, crypt, and md5 are one-way hashing algorithms whereas AES and DES are two-way encryption algorithms. AES and DES keys can be stored in the ICSF CKDS. However, the **dsconfig** utility does not generate these keys. See 5.7, “Encryption and Hashing” on page 132 for more information about AES and DES keys, and the encryption and hashing algorithms.

Note: If you are planning on having users bind or authenticate to the directory using the CRAM-MD5 or DIGEST-MD5 authentication mechanisms, the passwords must be un-encrypted, and then AES or DES encrypted to allow these bind mechanisms to properly work. See 5.2, “Authentication mechanisms supported by IBM Tivoli Directory Server for z/OS” on page 92 for more information about the CRAM-MD5 and DIGEST-MD5 authentication mechanisms.

2.4.2 Configuring for advanced replication and LDAP password policy

In z/OS V1R11, IBM Tivoli Directory Server for z/OS added support for advanced replication that allows separate replication topologies to be configured. This feature requires that the CDBM back end be configured. When using the **dsconfig** utility, the **CDBM_USEADVANCEDREPLICATION** parameter in the **ds.profile** must be set to on and the **SERVERCOMPATLEVEL** in the **ds.slapped.profile** must be 5 or greater. See 1.4, “Server Architectures” on page 5 for information about the supported advanced replication topologies.

In z/OS V1R12, IBM Tivoli Directory Server for z/OS introduced the LDAP password policy, which enforces an organization's password policy when passwords are stored in the **userPassword** attribute value. This feature requires that the CDBM back end be configured. When using the **dsconfig** utility, the **USEPASSWORDPOLICY** parameter in the **ds.profile** must be set to on and the **SERVERCOMPATLEVEL** in the **ds.slapped.profile** must be 6 or greater. See 5.6, “Password Policy” on page 127 for information about how to set up password policies in the IBM Tivoli Directory Server for z/OS.

2.4.3 Activity and audit logging

IBM Tivoli Directory Server for z/OS supports activity and audit logging of LDAP client operations in the LDAP server, which enables an LDAP administrator to keep a log of all connections to the LDAP server.

If planning on using the activity logging support in IBM Tivoli Directory Server for z/OS, verify that there is enough space to store the activity log for the LDAP client operations. The activity log can be stored in a partitioned dataset, sequential dataset, or a file in a z/OS UNIX System Services file system. In z/OS V1R12, automatic activity log archiving or rollover support was added that provides the ability to archive the current activity log file in a separate location (e.g. z/OS UNIX System Services directory) for load analysis and to help manage the size of the log file. Again, verify that there is enough space in this other location for storing these archived activity log files. See 13.3, “Using activity logging” on page 282 for more information about the features of the activity log support.

The audit log provides granular controls for the type of LDAP client activity (adds, searches, modifies, and so on) that should be audited in the server. The audit log can be configured to only log successful operations, error operations, or both. See 13.5, “Audit logging” on page 285 for more information about auditing.

2.4.4 Using the dsconfig utility

After deciding which back ends to configure or features to enable in your LDAP server, you are now probably ready to configure your own LDAP server. To help simplify the configuration of your own LDAP server, the **dsconfig** utility is provided. The **dsconfig** utility has a number of input files (**ds.profile**, **ds.slapped.profile**, **ds.racf.profile**, and **ds.db2.profile**) that

require input from one or multiple administrators (LDAP, System, Security, and Database) depending on the back ends or features that are to be enabled in your new LDAP server.

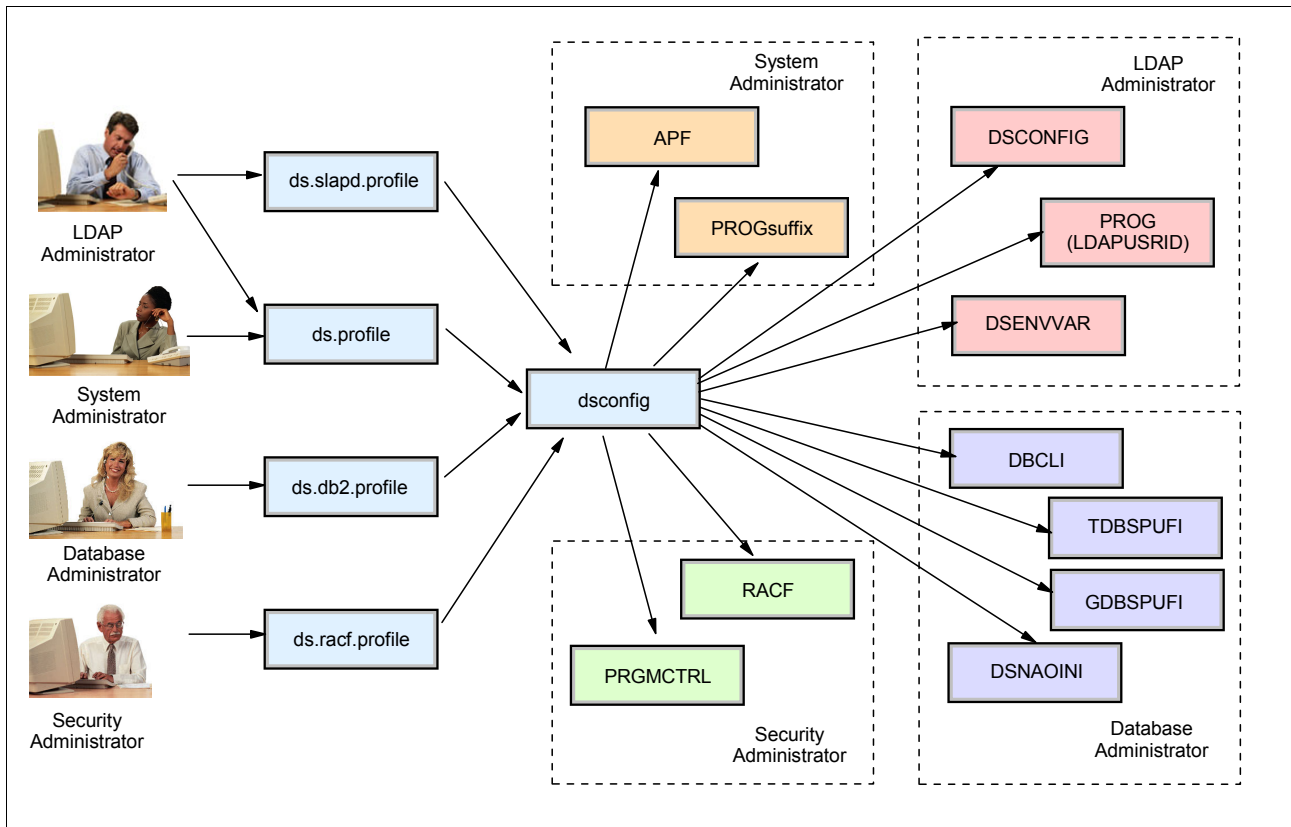


Figure 2-1 Output from the `dsconfig` utility

As illustrated in Figure 2-1, the output from the `dsconfig` utility generates JCL jobs, configuration files, and the LDAP server started task procedure.

Before running the `dsconfig` utility and updating the input files, copy the `ds.profile`, `ds.slapd.profile`, `ds.racf.profile`, and `ds.db2.profile` files from the `/usr/lpp/lldap/etc` directory to another location so that the files can easily be modified. The main input configuration file is `ds.profile`, and the `ds.slapd.profile`, `ds.racf.profile`, and `ds.db2.profile` files are referred to as advanced configuration files. These input files have an explanation of all required and optional parameters.

The `ds.profile` is the first file that must be updated because this profile specifies the `dsconfig` output dataset, LDAP administrator, the LDAP administrator's password, the back ends that are to be configured, the user ID that the LDAP server will run under, and the group that the user ID belongs to. Most parameters in this file are required. The location of the other profiles pointed to by the `SLAPD_PROFILE`, `DB2_PROFILE`, and `RACF_PROFILE` parameters need to be updated to point to the new directory location.

```
SLAPD_PROFILE = /usr/lpp/lldap/etc/ds.slapd.profile
DB2_PROFILE  = /usr/lpp/lldap/etc/ds.db2.profile
RACF_PROFILE = /usr/lpp/lldap/etc/ds.racf.profile
```

When the LDAP server is configured with the `dsconfig` utility, the LDAP server's user ID is granted CONTROL access to the UNIXPRIV.FILESYS profile. This gives the LDAP server's user ID super user authority to create the necessary files and directories required for the schema, file-based GDBM back end, and CDBM back end.

If you do not want to grant the LDAP server's user ID this file system authority, then you must verify the following:

1. If a value is specified for the SCHEMAPATH, GDBM_DATABASEDIRECTORY, or LDBM_DATABASEDIRECTORY parameters, the LDAP server's user ID must have the authority to create the directory if it does not exist. If the directory exists, the LDAP server's user ID must have read and write authority to the directory to create the necessary files.
2. If a value is not specified for the SCHEMAPATH parameter, the LDAP server's user ID must have the authority to create the /var/ldap/schema directory if it does not already exist, and to read and write to that directory if it does already exist.

The `ds.slapd.profile` has sections (global, environment, SSL/TLS, Kerberos, plug-in, and back ends), that are used as input to create the LDAP server configuration file (DSCONFIG) and the LDAP environment variables file (DSENVVAR). Most parameters in this profile are optional. However, if a configuration option needs something other than the default value, this file should be updated. This file should also be studied to verify that the default values for the LDAP server configuration file options are appropriate.

The `ds.db2.profile` is used by the DB2 administrator when configuring a DB2-based back end (TDBM or GDBM). This file allows the DB2 administrator to update the sizes of the DB2 tables and indicate which DB2 buffer pools to use. The parameters in this input file are all pre-filled in with default values that are useful for the majority of TDBM and GDBM back end users. When the TDBM back end is configured, the DBCLI, TDBSPUFI, and DSNAOINI files are created in the `dsconfig` output dataset. When the DB2-based GDBM back end is configured, the DBCLI, GDBSPUF and DSNAOINI files are created in the `dsconfig` output dataset.

The `ds.racf.profile` is used by the System administrator to set the LDAP server user ID's UID value and the GID value for the group ID. This file is used as input for the creation of the RACF job.

After updating the input files, run the `dsconfig` utility:

```
/usr/lpp/ldap/sbin/dsconfig -i ds.profile
```

The `dsconfig` utility generates the following members in the partitioned dataset specified by the OUTPUT_DATASET parameter in `ds.profile`:

- ▶ APF: A JCL job that sets APF authorization on libraries used by the LDAP server.
- ▶ *PROGSuffix*: A job run to grant APF authorization to datasets and libraries used by the LDAP server.
- ▶ DSCONFIG: The LDAP server configuration file.
- ▶ PROC (LDAPUSRID): Used to start the LDAP server as a started task. The name of the LDAP server started task procedure is the name of the LDAP user ID specified on the LDAPUSRID parameter in `ds.profile`.
- ▶ DSENVVAR: The LDAP server environment variables file.
- ▶ DBCLI: A JCL job that binds the CLI packages to DB2 and the DSNACLI plan.
- ▶ TDBSPUFI: A set of DB2 SQL statements, to be executed using the SPUFI tool, that defines database tables for the TDBM back end.
- ▶ GDBSPUFI: A set of DB2 SQL statements, to be executed using the SPUFI tool, that defines database tables for the DB2-based GDBM back end.
- ▶ DSNAOINI: The DB2 DSNAOINI initialization file.
- ▶ RACF: A JCL job that updates RACF to allow the LDAP server to run as a started task.
- ▶ PRGMCTRL: A JCL job that sets program control on libraries used by the LDAP server.

Now it is time to copy the members from the **dsconfig** output dataset and submit any jobs created:

1. Copy the LDAP server started task procedure to the target system's procedure library.
2. Copy `PROGsuffix` (where *suffix* is specified on the `PROG_SUFFIX` parameter in the `ds.profile` file) to the target system's PARMLIB.
3. Submit the following generated JCL in the following order:
 - a. RACF
 - b. APF
 - c. DBCLI (when TDBM or DB2-based GDBM is configured)
 - d. PRGMCTRLIf `TDBSPUFI` and `GDBSPUFI` were generated, submit them in `SPUFI` to create the tables for the TDBM or DB2-based GDBM back ends.
4. The LDAP server can now be started from the operator's console or SDSF:
 - From the operator's console, type: `s userid`
 - From SDSF, type: `/s userid`

Now your server should be started.

See “Configuring an LDAP server using the `dsconfig` utility” chapter in the *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for more information about the **dsconfig** utility.



Part 2

Concepts

In this part we discuss the concepts and features of the IBM Tivoli Directory Server for z/OS.



Back ends

This chapter discusses IBM Tivoli Directory Server for z/OS back ends.

3.1 Back end overview

A back end is a part of the LDAP server that administers a set of directory entries. Back ends differ in the type of directory entries they can store and where they store the entries.

IBM Tivoli Directory Server for z/OS provides a number of back ends, some of which are general purpose and some of which are used for more specific purposes. Each back end is described briefly below, with more details in the following sections of this chapter:

- ▶ **TDBM: General purpose directory**
 - Full LDAP V3 support
 - Data stored in DB2
 - Full scalability
 - Useful for medium to large directories
- ▶ **LDBM: General purpose directory**
 - Full LDAP V3 support
 - Data stored in z/OS UNIX System Services file system
 - Useful for small to medium sized directories
- ▶ **CDBM: Configuration directory**
 - Used to store dynamic configuration data
 - Data stored in z/OS UNIX System Services file system
- ▶ **SDBM: RACF users, groups, user-group connections, and general resource profiles**
 - Provides remote RACF administration and authentication
 - Data stored in RACF database
 - Fixed schema
 - Limited search capability: limited number of search filters that are supported in the SDBM back end (i.e. objectclass=*)
- ▶ **GDBM: Change log directory**
 - Contains records of changes to other back ends and RACF
 - Data stored in DB2 or z/OS UNIX System Services file system
 - Similar to LDBM/TDBM, but restricted operations
- ▶ **EXOP: Extended Operation directory**
 - Provides support for extended operations that retrieve z/OS Policy Director data
 - Not required for any other extended operations
- ▶ **Schema**
 - Single server-wide schema used by all back ends that simplifies administration of server

Each of the IBM Tivoli Directory Server for z/OS back ends can exploit various features provided by the LDAP server. Table 3-1 maps each back end to features that they can exploit:

Table 3-1 Back end table

Feature	TDBM	LDBM	CDBM	SDBM	GDBM	EXOP	Schema
Advanced Replication	x	x	x				x

Feature	TDBM	LDBM	CDBM	SDBM	GDBM	EXOP	Schema
Aliases	x	x	x				
Attribute Encryption	x	x	x				
Basic Replication	x	x					
Bulk Load	x						
Bulk Unload	x	x	x				x
Change logging	x	x	x				x
Multi-Server Operational Modes	x	x	x		x		
Native Authentication	x	x	x				
Password Policy	x	x	x				
Policy Director						x	
RACF Administration				x			
Referrals	x	x					

Note: The change log is implemented in the GDBM back end.

The schema back end is used by all other back ends.

The features provided in Table 3-1 on page 30 might not be exhaustive. For a full explanation of the support provided by IBM Tivoli Directory Server for z/OS, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

The IBM Tivoli Directory Server for z/OS LDAP features that are listed in the table above are each briefly described below:

- ▶ **Advanced Replication:** The LDAP server supports advanced replication of specified subtrees in a TDBM, LDBM, or CDBM back end to other servers. Advanced replication provides four types of server topologies for deploying an advanced replication environment.
- ▶ **Aliases:** Alias support provides a means for a TDBM, LDBM, or CDBM directory entry to point to another entry in the same directory.
- ▶ **Attribute Encryption:** The LDAP server supports encryption of the values of several critical attributes to prevent unauthorized access to these attribute values in TDBM, LDBM, or CDBM back ends. The following attributes can be encrypted using AES or DES: **secretKey**, **replicaCredentials**, **ibm-replicaKeyPwd**, and **ibm-slappedMasterPw**. The **userPassword** attribute can also be encrypted using AES, crypt, DES, MD5, SHA, and Salted SHA (SSHA).
- ▶ **Basic Replication:** The LDAP server supports basic replication of TDBM and LDBM directory data to other servers. Basic replication provides two types of server topologies for deploying a basic replication environment.
- ▶ **Bulk Load:** The LDAP server can load a large number of entries in LDIF format into a TDBM back end directory using the **ldif2ds** utility.
- ▶ **Bulk Unload:** The LDAP server can unload a large number of entries from a TDBM, LDBM, or CDBM back end or the schema using the **ds2ldif** utility into a file in LDIF format.

- ▶ **Change logging:** Information about a change to a TDBM, LDBM, CDBM, or schema entry (cn=schema), or to an object controlled by an application (for example, a RACF user, group, user-group connection, or general resource profile) entry, can be saved in a change log entry.
- ▶ **Multi-Server Operational Modes:** Multiple concurrent instances of LDAP servers can use a given TDBM, LDBM, CDBM, or GDBM database to store directory data.
- ▶ **Native Authentication:** The LDAP server has the ability to authenticate a user with the Security Server using a TDBM, LDBM, or CDBM back end. The user specifies a Security Server password or password phrase on a simple bind to the back end. The TDBM, LDBM, or CDBM entry that contains the bind DN contains the RACF ID, which is then used with the specified password or password phrase to authenticate the user with z/OS Security Server.
- ▶ **Password Policy:** Password policy rules can be applied to TDBM, LDBM, or CDBM entries that have a **userPassword** value.
- ▶ **Policy Director:** The EXOP back end provides support for extended operations that retrieve z/OS Policy Director data.
- ▶ **RACF Administration:** The SDBM back end allows a RACF administrator to remotely manage the RACF database using the LDAP protocol.
- ▶ **Referrals:** A referral entry can be added to a TDBM or LDBM back end to indicate that the back end does not contain that entry or any entries below it and to identify another LDAP server that might contain those entries.

3.2 TDBM back end

The z/OS LDAP server provides the TDBM back end as a general purpose directory, meaning that any type of directory information can be stored within it. The TDBM back end uses IBM Database 2 (DB2), a powerful and scalable database product, for its data storage facility. The TDBM back end is therefore highly scalable itself and designed to handle medium to large directories.

Although every installation has differing limitations, the general recommendation is that TDBM should be used if the directory size is expected to scale beyond the maximum recommended sizes for LDBM directories. IBM Tivoli Directory Server for z/OS recommends a maximum of 250,000 entries in 31-bit mode and a maximum of 500,000 entries in 64-bit mode for LDBM. If a directory is expected to scale beyond these sizes, then TDBM is recommended. Note that TDBM is provided in 31-bit mode only.

Advantages

- ▶ TDBM is a general purpose directory.
- ▶ TDBM is highly scalable and backed by DB2.
- ▶ TDBM exploits DB2 data sharing and can be shared in a Parallel Sysplex®.
- ▶ Multiple TDBM back ends can be configured.

Considerations

- ▶ TDBM requires DB2, which is more complex to set up than a file-based back end.
- ▶ The LDAP server must be started in 31-bit mode if using TDBM.

3.2.1 TDBM configuration

The LDAP server can be configured with a TDBM back end using the **dsconfig** utility to automate the process, or manually for situations where the **dsconfig** utility is not adequate or further configuration updates are required.

Automated configuration

The LDAP configuration utility, **dsconfig**, can simplify and automate the LDAP server configuration process for TDBM (among other back ends). For TDBM configuration specifically, **dsconfig** will generate the following members:

- ▶ **DBCLI**: A JCL job that binds the CLI packages to DB2 and the DSNACLI plan.
- ▶ **DSNAOINI**: The DB2 CLI initialization file.
- ▶ **TDBSPUFI**: A set of DB2 SQL statements, to be executed using the SPUFI tool, that defines database tables for the TDBM back end.

After DB2 has been started, the DBCLI JCL can be submitted to bind the CLI packages to DB2 and the DSNACLI plan. The TDBSPUFI member can then be submitted through the DB2 SPUFI interactive tool, to define database tables for the TDBM back end.

The **dsconfig** utility can also produce a number of other files, including the DSCONFIG and DSENVVAR LDAP server configuration and environment variables files, and a procedure member needed to start the LDAP server as a started task. Using these files, along with the DSNAOINI CLI initialization file, the LDAP server can be started with a TDBM back end.

For a complete step by step process of using the **dsconfig** utility, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

Manual configuration

The TDBM back end can be configured manually if the **dsconfig** utility is not adequate or if further configuration updates are required. A roadmap for configuring a TDBM back end can be found in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05. After DB2 is installed and set up for TDBM, the LDAP server is ready to be configured and started.

The TDBM back end-specific section of the `ds.conf` configuration file contains configuration options that are specific to the TDBM back end. It is possible to have one or more of these sections depending on the number of TDBM back ends an installation will use. To configure the LDAP server to run with a TDBM back end, the configuration files can first be copied from the `/usr/lpp/lldap/etc` directory. Figure 3-1 shows a sample TDBM back end-specific section of the `ds.conf` configuration file:

```
# TDBM back end section
database tdbm GLDBTD31
suffix "o=Your Company"
dbuserid GLDSRV
```

Figure 3-1 TDBM back end section

Note that GLDSRV is the TDBM database owner and `o=Your Company` can be any valid DN suffix.

The database, suffix, and dbuserid configuration options are required for configuration of a TDBM back end.

The following options are optional for configuring a TDBM back end: `ac1SourceCacheSize`, `attrOverflowCount`, `attrOverflowSize`, `changeLoggingParticipant`, `dnToEidCacheSize`, `dsnaoini`, `entryCacheSize`, `entryOwnerCacheSize`, `extendedGroupSearching`, `filterCacheBypassLimit`, `filterCacheSize`, `include`, `persistentSearch`, `readonly`, `serverName`, `sizeLimit`, and `timeLimit`.

For more details regarding configuration options, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

Depending on the type of entries intended to be added to the TDBM back end, the LDAP server schema might need to be modified to add the required attributes and objectclasses.

3.2.2 Porting TDBM data from IBM Tivoli Directory Server for z/OS to IBM Tivoli Directory Server for z/OS

If an existing IBM Tivoli Directory Server for z/OS TDBM database needs to be copied to a new IBM Tivoli Directory Server for z/OS TDBM database, the `ds21dif` utility can be used to unload the existing database into an LDIF file, and either the `ldif2ds` or `ldapadd` utilities can be used to load these entries into the new database. If the load is for more than 100,000 directory entries, use `ldif2ds`.

If porting entries to a TDBM database that is not on the same LDAP server as the existing TDBM database, the target LDAP server schema might require updates. Prior to porting entries, the target LDAP server schema might need to be updated to contain the attributes and object classes that were in use by the TDBM entries on the existing server. The `ds21dif` or `ldapsearch` utilities can be used to unload the LDAP server schema from the source LDAP server and the `ldapmodify` utility can then be used to load the schema LDIF file into the target LDAP server.

3.2.3 Porting TDBM data from ISS to IBM Tivoli Directory Server for z/OS

If an existing ISS TDBM database needs to be copied to a new IBM Tivoli Directory Server for z/OS database, the DB2 copy utility can be used. The DDL specifications defined for the existing ISS database must be used. Otherwise, the DB2 copy utility will produce unreliable results.

Note: In IBM Tivoli Directory Server for z/OS V1.11, the default SPUFI scripts increase the DN_TRUNC column size from 32 to 64 bytes in the DIR_ENTRY table. If you need to increase the size of the DN_TRUNC column or make other customization to the IBM Tivoli Directory Server for z/OS DDL, you must unload the ISS TDBM database to LDIF format. The DB2 utilities cannot be used to copy an old ISS TDBM database from a prior z/OS release to a IBM Tivoli Directory Server for z/OS TDBM database on a later z/OS release. The copy must be made from an ISS TDBM database to a IBM Tivoli Directory Server for z/OS TDBM database that are both running on the same z/OS release level.

Use the following steps to create a new IBM Tivoli Directory Server for z/OS TDBM database from an existing ISS TDBM database:

1. Create the IBM Tivoli Directory Server for z/OS database using the DDL specifications defined for the existing ISS TDBM database.
2. Copy the ISS TDBM database into the new IBM Tivoli Directory Server for z/OS TDBM database using the DB2 copy utility. Note that the MISCTS table space is a segmented

table space and the RESUME option must be specified when loading this table space from data unloaded from the ISS TDBM database.

The target IBM Tivoli Directory Server for z/OS server schema might require updates for the entries being ported. Steps for updating the schema when migrating TDBM from an ISS LDAP server to a IBM Tivoli Directory Server for z/OS LDAP server can be found in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

3.2.4 Using the TDBM back end

The following example adds an entry to a TDBM back end:

Create a file, joe.add, containing the entry to be added as shown in Example 3-1.

Example 3-1 Entry to be loaded

```
dn: cn=Joe Johnson,ou=users,dc=yourcompany,dc=com
objectclass: person
sn: Johnson
userpassword: joespw
```

Invoke the `ldapadd` utility to add the entry:

```
ldapadd -h 127.0.0.1 -p 389 -D cn=admin -w secret -f joe.add
```

The following example modifies an entry in a TDBM back end:

Create a file, joe.mod, containing the changes as shown in Example 3-2.

Example 3-2 Entry to modify

```
dn: cn=Joe Johnson,ou=users,dc=yourcompany,dc=com
add: telephonenumber
telephonenumber: 123-456-7890
-
replace: userpassword
userpassword: joesnewpw
```

Invoke the `ldapmodify` utility to modify the entry:

```
ldapmodify -h 127.0.0.1 -p 389 -D cn=admin -w secret -f joe.mod
```

The following example searches for an entry in a TDBM back end:

Invoke the `ldapsearch` utility to display all entries with `sn=Johnson`:

```
ldapsearch -h 127.0.0.1 -p 389 -D cn=admin -w secret -b dc=yourcompany,dc=com
sn=Johnson
```

The search result is shown in Figure 3-2 on page 36.

```

cn=Joe Johnson,ou=users,dc=yourcompany,dc=com
objectclass=person
objectclass=top
sn=Johnson
userpassword=joesnewpw
cn=Joe Johnson
telephonenumber=123-456-7890

```

Figure 3-2 Search result

The following example deletes an entry from a TDBM back end:

Invoke the `ldapdelete` utility to delete the entry:

```

ldapdelete -h 127.0.0.1 -p 389 -D cn=admin -w secret "cn=Joe
Johnson,ou=users,dc=yourcompany,dc=com"

```

The following example unloads data from a TDBM back end:

Invoke the 31-bit version of `ds2ldif` to unload all entries from the TDBM back end named `tdbm1` in the LDAP server configuration file `/etc/ldap/ds.conf`:

```

ds2ldif31 -j -o /tdbmdata/ldif.data -n tdbm1 -f /etc/ldap/ds.conf

```

The two entries that are in the TDBM back end are written to the file named `/tmpdata/ldif.data` as shown in Figure 3-3.

```

version: 1

dn: o=tdbm
objectclass: organization
objectclass: top
o: tdbm
ibm-entryuuid: 3A67E000-2E5C-1876-99D0-402064040959
aclentry: cn=Anybody:normal:rsc:system:rsc
aclpropagate: TRUE
entryowner: CN=ADMIN
ownerpropagate: TRUE

dn: cn=entry1,o=tdbm
objectclass: newPilotPerson
objectclass: person
objectclass: top
cn: entry1
sn: 1
uid: entry1
userpassword:: c2VjcmV0
ibm-entryuuid: 0C3DE000-F85D-1884-94B3-402084027431

```

Figure 3-3 ldif contents

The following example loads data into a TDBM back end:

Invoke the `ldif2ds` utility to load all of the entries from the input LDIF file into a TDBM database:

```

ldif2ds -cpl -i /tmpdata/ldif.data -o admin3.prv -d ERROR

```

This `ldif2ds` utility invocation checks, prepares, and loads the LDIF data from `/tmpdata/ldif.data`, uses the output datasets `ADMIN3.PRIV.BULKLOAD.INPUT.xxx` and `ADMIN3.PRIV.BULKLOAD.JCL`, and specifies a debug level of `ERROR`. It also uses the default server configuration file of `/etc/ldap/ds.conf`. The value of the `adminDN` option in the LDAP server configuration file is used as the default creator of each loaded entry.

3.2.5 Tuning the TDBM back end

The TDBM back end uses DB2 for storing directory data and takes advantage of caching to handle repetitive access to directory data. As with any LDAP back end, the server performs best when directory data is read frequently and updated less frequently. If directory data is frequently updated and non-cached data is randomly accessed, TDBM performance can degrade.

There are a few areas where tuning can help improve the performance of the TDBM back end:

- ▶ **Cache Tuning:** TDBM caches provide a significant benefit to performance, allowing the server to bypass read operations to the database.
- ▶ **DB2 tuning:** DB2 tuning ensures that TDBM efficiently accesses DB2, and that response times remain steady even while the database scales in size.
- ▶ **TDBM SQL tuning:** The choices made with the initial setup of the TDBM database will influence DB2 performance.
- ▶ **Configuration file tuning:** The TDBM back end section of the LDAP server configuration file will also affect DB2 performance.

Cache tuning

Caches are beneficial when the majority of directory operations read data. This allows the server to bypass read operations to DB2. Tuning cache sizes enables high percentage cache hit rates. The process of cache tuning involves monitoring cache effectiveness and adjusting cache sizes to increase their percent hit rate.

Monitoring of cache use should be performed during typical workloads. This can be done using either a `cn=monitor` search or the operator console `MODIFY` command to retrieve current cache statistics. The cache hit rate, the current number of entries, and the maximum allowed entries (the configured size) should be examined. The number of cache refreshes and the average size of the cache at refresh should also be noted. If the cache hit rate is well below 100% and the cache is frequently fully populated, consider increasing the cache size. Because cache changes require updates to configuration options, the server configuration file must be updated and the server must be restarted to update cache settings.

For more details about cache tuning refer to *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

DB2 tuning

The following tasks are related to DB2 tuning, and are crucial for improving TDBM performance. These tasks are typically performed by a database administrator:

- ▶ Periodically reorganize the database (REORG) and maintain database statistics (RUNSTATS)

The TDBM table spaces and indexes should occasionally be reorganized using the DB2 REORG utility. This process will help to improve DB2 access performance and will help to reclaim fragmented space.

The DB2 catalog should be updated occasionally with current statistics for the TDBM database, table spaces, tables, indexes, and partitions. This can be performed using the

RUNSTATS utility. This information enables DB2 to select efficient access paths to the TDBM database. Additionally, this information can be useful for database administrators to determine when the database should be reorganized. For suggested parameters to specify with the RUNSTATS utility, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

When a z/OS LDAP directory is initially updated with a large amount of data, followed by gradual growth, run the REORG and RUNSTATS utilities immediately after the directory is populated with the initial data, and before being rolled out to production.

If the initial directory population is performed by an application, as opposed to the **ldif2ds Load** utility, it might be necessary to run REORG and RUNSTATS one or more times during the initial directory population.

If the database is not periodically reorganized and statistics are not maintained, then poor access paths can be chosen, causing increased response times as the database size scales.

- ▶ Allocate DB2 buffer pools large enough to minimize I/O to the TDBM database

DB2 buffer pool allocations should be evaluated to make sure they are sufficient for the TDBM database. It is typically beneficial to isolate specific TDBM table spaces and indexes to their own buffer pools. Separating the indexes from the table spaces can help ensure that index buffers remain in the buffer pools. Additionally, this technique can help the database administrator to determine the overall TDBM usage of buffer pools when specific tables and indexes correspond to specific buffer pools. The DB2 Performance Monitor for z/OS can be beneficial for monitoring buffer pool activity.

If an installation has a large directory with frequent update operations on the directory, partitioning the DB2 tables can also increase performance. See Partitioning DB2 tables for TDBM in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for more information.

TDBM SQL tuning

A number of TDBM SQL choices can affect the performance of TDBM when accessing data within DB2:

- ▶ Tablespace LOCKSIZE

The default LOCKSIZE of ANY is generally sufficient if the majority of activity consists of read operations, with few database updates. This typically results in PAGE locking that causes locking of rows for directory entries other than the one being updated.

On the other hand, if the database is frequently updated, DB2 deadlocks can occur in the TDBM database with PAGE locking. In this case, LOCKSIZE ROW should be set on the TDBM tablespaces that contains the DIR_ENTRY and the DIR_SEARCH table.

- ▶ Size of DIR_ENTRY table's DN_TRUNC column

The DN_TRUNC column is used to index data in the DIR_ENTRY table and speed up retrieval of directory entries by their distinguished name (DN). This column holds the leading portion of each DN, and should be defined long enough to make most values unique.

Some applications generate directory entries where the leading portion of the DN is identical. For example, Tivoli Access Manager (TAM) generates entries under each user entry in the namespace where the DN starts with "cn=secPolicyData,secAuthority=Default,". To provide uniqueness, it is suggested that installations using TAM with the z/OS LDAP server define the DN_TRUNC column to be 64 bytes in length.

This column length should be defined correctly during the initial setup of the directory. If the length is changed, the DIR_ENTRY table must be redefined, and the directory must be unloaded and reloaded to implement the change.

- ▶ Size of DIR_SEARCH table's VALUE column

The VALUE column is used to index data in the DIR_SEARCH table and speed up retrieval of directory entries for search requests using the search filter values. This column holds the leading portion of textual attribute values, and should be defined long enough to accommodate most values specified in search filters. This column should not be made significantly larger than required because this can cause the DIR_SEARCH table and its index to increase substantially in size.

This column length should be defined correctly during the initial setup of the directory. If the length is changed, the DIR_SEARCH table must be redefined, and the directory must be unloaded and reloaded

Configuration file tuning

The **attrOverflowSize** can be modified in the TDBM back end section of the LDAP server configuration file.

- ▶ The **attrOverflowSize** configuration value

This configuration option specifies the threshold size of attribute values that are to be stored separately from the DIR_ENTRY data and are instead stored in the DIR_LONGATTR overflow table.

This option helps to avoid reading overflow data for searches that do not request the attribute. For example, if a directory contains an attribute with JPEG data and searches often do not request this attribute, then **attrOverflowSize** can avoid reading the data from the database.

This option value should be specified large enough so that data is typically retrieved from the DIR_ENTRY data. Note that entries with overflow data are not eligible for the entry cache, and so setting this value too small can impact search performance.

3.3 LDBM back end

The z/OS LDAP server provides the LDBM back end as a general purpose directory, meaning that any type of directory information can be stored within it. The LDBM back end uses a z/OS UNIX System Services file system for its data storage facility. The LDBM back end provides the same functionality as the TDBM back end, but is suitable for small to medium size directories.

Although every installation has differing limitations, use LDBM for a maximum of 250,000 entries in 31-bit mode and a maximum of 500,000 entries in 64-bit mode. If a directory should scale beyond these sizes, use the TDBM back end.

The LDBM back end keeps its entries in memory for quick access and requires a minimum amount of setup. When the LDAP server is not running, LDBM stores its directory information in z/OS UNIX System Services files.

Advantages

- ▶ LDBM is a general purpose directory.
- ▶ Entries are kept in memory while the server is running, enabling quick access.
- ▶ LDBM requires a minimal amount of setup.
- ▶ LDBM runs in both 31-bit and 64-bit modes.

- ▶ LDBM can be shared in a Parallel Sysplex.
- ▶ Multiple LDBM back ends can be configured.

Considerations

- ▶ LDBM has limitations on scalability.
- ▶ Large LDBM directories requires a lot of memory.
- ▶ Server start-up and shutdown can be time consuming.
- ▶ Paging hurts LDBM performance.

3.3.1 LDBM configuration

IBM Tivoli Directory Server for z/OS can be configured with an LDBM back end by either using the **dsconfig** utility to automate the process, or configuration can be performed manually for situations where the **dsconfig** utility is not adequate or further configuration updates are required.

Automated configuration

The LDAP configuration utility, **dsconfig**, can simplify and automate the LDAP server configuration process for LDBM (among other back ends). There are no members generated specifically for LDBM. However, there are general use members that are generated and help to deploy an LDAP server with LDBM.

The **dsconfig** utility produces a number of files, including the DSCONFIG and DSENVVAR LDAP server configuration and environment variables files, and a procedure member needed to start the LDAP server as a started task. Using these files, the LDAP server can be started with an LDBM back end.

For a complete step by step process of using the **dsconfig** utility, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS, SC23-5191-05*.

Manual configuration

The LDBM back end can be configured manually if the **dsconfig** utility is not adequate or if further configuration updates are required. A roadmap for configuring an LDBM back end can be found in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS, SC23-5191-05*. After a z/OS UNIX System Services file system is set up for LDBM, the LDAP server is ready to be configured and started.

The LDBM back end-specific section of the `ds.conf` configuration file contains configuration options that are specific to the LDBM back end. It is possible to have one or more of these sections depending on the number of LDBM back ends an installation will use. To configure the LDAP server to run with an LDBM back end, the configuration files can first be copied from the `/usr/lpp/ldap/etc` directory. Figure 3-4 is a sample LDBM back end-specific section of the `ds.conf` configuration file.

```
# LDBM back end section
database ldbm GLDBLD31/GLDBLD64
suffix "o=Your Company"
```

Figure 3-4 LDBM back end section

Note: Although the example uses `o=Your Company` as the suffix, the suffix can be any valid DN suffix.

The database and suffix configuration options are required for configuration of an LDBM back end.

The following options are optional for configuring an LDBM back end: **attrOverflowCount**, **changeLoggingParticipant**, **commitCheckpointEntries**, **commitCheckpointTOD**, **databaseDirectory**, **extendedGroupSearching**, **fileTerminate**, **filterCacheBypassLimit**, **filterCacheSize**, **include**, **persistentSearch**, **readOnly**, **sizeLimit**, and **timeLimit**.

For more details regarding configuration options, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

Depending on the type of entries intended to be added to the LDBM back end, the LDAP server schema might need to be modified to add the required attributes and objectclasses.

3.3.2 Porting LDBM data

If an existing LDBM database needs to be copied to a new LDBM database, the **ds2ldif** utility can be used to unload the existing database into an LDIF file, and the **ldapadd** utility can be used to load these entries into the new database.

If porting entries to an LDBM database that is not on the same LDAP server as the existing LDBM database, the target LDAP server schema might require updates. Prior to porting entries, the target LDAP server schema might need to be updated to contain the attributes and object classes that were in use by the LDBM entries on the existing server. The **ds2ldif** or **ldapsearch** utilities can be used to unload the LDAP server schema from the source LDAP server and the **ldapmodify** utility can then be used to load the schema LDIF file into the target LDAP server.

3.3.3 Creating a sample server with an LDBM back end

IBM Tivoli Directory Server for z/OS ships an example that provides a set of instructions for properly configuring an IBM Tivoli Directory Server for z/OS LDAP server with an LDBM back end. The example files are located in `/usr/lpp/ldap/examples/sample_server`. The `ds.README` file provides step-by-step instructions for getting an LDAP server configured and started. The `sample.ldif` file contains a set of directory entries that can be added to the LDAP server's LDBM back end.

3.3.4 Using the LDBM back end

Section 3.2.4, "Using the TDBM back end" on page 35 contains examples that can also be used for an LDBM back end. See the examples for adding, modifying, searching, and deleting an entry, and the example for unloading data. Note that for bulk loading data, the **ldif2ds** utility can only be used for loading TDBM entries. LDBM entries can be loaded using the **ldapadd** utility.

3.3.5 Tuning the LDBM back end

The LDBM back end uses a z/OS UNIX System Services file system for persistent storage of directory data. When the LDAP server is executing, the entire directory contents are held within its address space, including index structures for quick access.

The storage of entries in memory while the server is executing provides quick access to directory data, because LDAP operations that read directory data involve no file I/O. LDAP

operations that update the directory typically perform file I/O only when writing the changed data to the LDBM checkpoint file. Additionally, index updates only occur within memory in the z/OS LDAP server's address space.

The LDBM back end naturally runs into resource issues as a directory scales in size. The resources affected are listed below, with usage typically proportional to directory size:

- ▶ Memory considerations: The memory required within the LDAP server address space.
- ▶ LDAP server initialization time: The LDAP server initialization time, both elapsed time and processor time.
- ▶ LDAP database commit processing: The time required for the LDAP server to commit the directory.
- ▶ File system space: The file system space required to store the persistent directory data.

Memory Considerations

Because the entire LDBM directory is kept in memory in the LDAP address space, plans should be made accordingly. The amount of memory required can be estimated based on the size of the LDIF data used to load the directory. For 31-bit mode, the memory required to store the data is about 7 to 10 times the size of the LDIF file. For 64-bit mode, the memory required to store the data is as approximately 10 to 15 times the size of the LDIF file.

These are estimates only. Additionally, these estimates only pertain to the memory required to hold the LDBM directory. Plans must also account for the storage required to run the z/OS LDAP server.

If the z/OS LDAP server is run in multi-server mode, every LDAP server that shares the LDBM directory will also store the entire LDBM directory in memory within their address space. Therefore, each of these servers will require approximately the same amount of memory as the LDAP master server.

Use the TDBM back end for systems with storage constraints or large directories.

LDAP server initialization time

When the LDAP server is started with an LDBM back end, LDBM directory is read into memory and the necessary index structures are built. The index structures are used to increase the performance of search processing. Server startup with LDBM can be time consuming, taking upwards of several minutes. This startup time is dependent on a number of factors, including the speed of the processor and DASD, and the current system workload. The total startup time is generally proportional to the size of the directory.

As mentioned previously, when the z/OS LDAP server is run in multi-server mode, every LDAP server that shares the LDBM directory also stores the entire LDBM directory in memory within their address space. Although replica servers require nearly the same amount of time as the master server to read the directory into memory, the time required to start up the replica is typically longer than that of the master. Initialization of a sysplex replica also consumes processor time on the sysplex master while the master sends data to the replica. The processor time consumed by sysplex master is estimated to be approximately one-third that consumed by the replica during startup.

LDAP database commit processing

The LDBM directory is stored in database files and the checkpoint file within a z/OS UNIX System Services file system. There is an LDBM database file for each suffix defined in the back end, and a single checkpoint file for the entire back end. The database files contain the entire directory up to the last database commit point, whereas the checkpoint file contains the directory updates made since the last commit point.

When commit processing occurs, the checkpoint files are used to refresh the database files. Commit processing occurs at the following times:

- ▶ When the number of checkpoint entries exceeds the **commitCheckpointEntries** value in the LDAP server configuration file.
- ▶ When the time of day reaches the **commitCheckpointTOD** value in the LDAP server configuration file.
- ▶ When the LDAP server COMMIT operator command is invoked.
- ▶ When the LDAP server is shut down normally.
- ▶ When the LDAP server is restarted and uncommitted updates exist in the checkpoint file after an abnormal termination of the LDAP server.

The processor and file I/O resources required for commit processing are also proportional to the directory size. Commit processing can take a minute or more for larger directories, depending on resource competition.

During commit processing, copies of the database files and the checkpoint file are made, and file system space should account for this. Therefore, enough file system space should be available to accommodate two copies of each directory file in addition to the maximum size of the checkpoint file. The amount of file system space required for the checkpoint file is best determined through experimentation because it depends on the updates that are being performed.

Another impact to account for is that update requests from clients will not be processed during commit processing. Therefore the use of the **commitCheckpointEntries** configuration option is not recommended because the timing of these commits cannot be scheduled. Use the following methods because they can be scheduled at a time when client update requests are minimized:

- ▶ Use the **commitCheckpointTOD** configuration option
- ▶ Automate the use of the LDAP server COMMIT operator command
- ▶ Use planned shutdowns of the LDAP server to control when commit processing occurs

File system space

The amount of space required to store an LDBM back end in a z/OS UNIX System Services file system can be estimated at four to six times the size of the input LDIF data. This is calculated based on the space required to hold the LDBM back end data, which is generally two to three times the size of the input LDIF data, in addition to the space required for commit processing. During LDBM commit processing, each of the LDBM back end files is copied, therefore doubling the amount of required file system space until commit processing completes.

3.3.6 Sample LDBM benchmark data

The following data was gathered from benchmarks using an LDBM database on z/OS UNIX System Services using ESS800 model 2105 DASD, and running on a z9® model 2094 processor with no competing applications running.

Database size	500,000 entries
LDIF file size	590 megabytes
LDAP server storage	5464 megabytes
Sysplex master initialization elapsed time	121 seconds
Sysplex master initialization processor time	98 seconds

Sysplex replica initialization elapsed time	180 seconds
Sysplex replica initialization processor time	103 seconds
Sysplex master processor time consumed by replica initialization	34 seconds
LDAP server database commit elapsed time	50 seconds
LDAP server database commit processor time	43 seconds

3.4 CDBM back end

The z/OS LDAP server provides the CDBM back end to store configuration information. The CDBM back end uses a z/OS UNIX System Services file system for its data storage facility.

The CDBM back end is used by the z/OS LDAP server to store configuration information for features such as advanced replication and password policy support. It provides dynamic access to configuration information, as opposed to the `ds.conf` configuration file that can only be read when the LDAP server is started.

When the LDAP server is started with CDBM configured for the first time, the server creates the following entries:

- ▶ `cn=ibmpolicies`
- ▶ `cn=pwdpolicy,cn=ibmpolicies` (if server compatibility level is 6 or greater)
- ▶ `cn=configuration`
- ▶ `cn=Replication,cn=configuration`
- ▶ `cn=Log Management,cn=configuration`
- ▶ `cn=Replication,cn=Log Management,cn=configuration`

The CDBM suffix entries are generated automatically by the LDAP server and cannot be loaded by a client application.

Advantages

- ▶ Configuration information stored in the CDBM back end can be searched and modified while the LDAP server is running.
- ▶ Entries are kept in memory while the server is running, enabling quick access.
- ▶ CDBM requires a minimal amount of setup.
- ▶ CDBM runs in both 31-bit and 64-bit modes.
- ▶ CDBM can be shared in a Parallel Sysplex.

Considerations

- ▶ The CDBM back end has similar disadvantages as the LDBM back end. However, many of the disadvantages are never encountered with a CDBM directory because CDBM typically contains only a small number of configuration entries.

3.4.1 CDBM Configuration

IBM Tivoli Directory Server for z/OS can be configured with a CDBM back end by using the `dsconfig` utility to automate the process, or manually for situations where the `dsconfig` utility is not adequate or further configuration updates are required.

Automated configuration

The LDAP configuration utility, **dsconfig**, can simplify and automate the server configuration process for CDBM (among other back ends). There are no members generated specifically for CDBM. However, there are general use members that are generated and help to deploy an LDAP server with CDBM.

The **dsconfig** utility produces a number of files, including the DSCONFIG and DSENVVAR LDAP server configuration and environment variables files, and a procedure member needed to start the LDAP server as a started task. Using these files, the LDAP server can be started with a CDBM back end.

For a complete step by step process of using the **dsconfig** utility, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

Manual configuration

The CDBM back end can be configured manually if the **dsconfig** utility is not adequate or if further configuration updates are required. A roadmap for configuring a CDBM back end can be found in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05. After a z/OS UNIX System Services file system is set up for CDBM, the LDAP server is ready to be configured and started.

The CDBM back end-specific section of the `ds.conf` configuration file contains configuration options that are specific to the CDBM back end. Only one CDBM back end can be configured in an LDAP server.

To configure the LDAP server to run with a CDBM back end, the configuration files can first be copied from the `/usr/lpp/lap/etc` directory. Figure 3-5 is a sample CDBM back end-specific section of the `ds.conf` configuration file.

```
#CDBM back end section
database CDBM GLDBCD31/GLDBCD64
```

Figure 3-5 CDBM back end section

The database configuration option is required for configuration of a CDBM back end.

The following options are optional for configuring a CDBM back end: **attrOverflowCount**, **changeLoggingParticipant**, **commitCheckpointEntries**, **commitCheckpointTOD**, **databaseDirectory**, **extendedGroupSearching**, **fileTerminate**, **filterCacheBypassLimit**, **filterCacheSize**, **include**, **persistentSearch**, **readOnly**, **sizeLimit**, and **timeLimit**.

For more details regarding configuration options, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

Depending on the type of entries intended to be added to the CDBM back end, the LDAP server schema might need to be modified to add the required attributes and objectclasses.

The server compatibility level must be 5 or greater for the CDBM back end to be configured. The CDBM back end cannot be used in a Parallel Sysplex containing mixed levels of z/OS LDAP servers. In a Parallel Sysplex such as this, the server compatibility level must be less than 5.

3.4.2 Using the CDBM back end

The following example searches the `cn=Replication,cn=configuration` suffix:

Display the `cn=Replication,cn=configuration` entry:

```
ldapsearch -D cn=admin -w secret -s base -b cn=Replication,cn=configuration
objectclass=*
```

The command returns the data shown in Figure 3-6 on page 46.

```
cn=Replication,cn=configuration
objectclass=ibm-slapdReplicationConfiguration
objectclass=top
cn=Replication
ibm-slapdmaxpendingchangesdisplayed=200
ibm-slapdreplcontextcachesize=100000
ibm-slapdreplmaxerrors=0
ibm-slapdreplconflictmaxentrysize=0
ibm-replicationonhold=FALSE
```

Figure 3-6 Search results for replication configuration query

The following example modifies an attribute:

In this case we modify the `ibm-replicationOnHold` attribute in the `cn=Replication,cn=configuration` entry.

Create a file, `replconfig.mod`, containing the changes that will suspend replication for all replication agreements in the server as shown in Figure 3-7.

```
dn: cn=Replication,cn=configuration
replace: ibm-replicationOnHold
ibm-replicationOnHold: TRUE
```

Figure 3-7 Modifying entry

Invoke the `ldapmodify` utility to modify the entry:

```
ldapmodify -h 127.0.0.1 -p 389 -D cn=admin -w secret -f replconfig.mod
```

3.4.3 Tuning the CDBM back end

If the CDBM back end is only used to store configuration entries, then no tuning is necessary. However, if the CDBM back end is used to store user defined entries, refer to 3.3.5, “Tuning the LDBM back end” on page 41 for performance considerations.

3.5 SDBM back end

The SDBM back end allows a RACF administrator to remotely manage the RACF database using the LDAP protocol. This allows the RACF administrator to be anywhere in the world and able to remotely administer the RACF database anywhere there is an LDAP client. The SDBM back end provides these features:

- ▶ Authentication with RACF users.

- ▶ Add, modify, and delete RACF users, groups, and general resources. Note that dataset resources are not supported.
- ▶ Add, modify, and delete user connections to groups.
- ▶ Add and remove users and groups in general resource access lists.
- ▶ Modify SETROPTS options that affect classes (for example, RACLIST).
- ▶ Retrieve RACF information for users, groups, connections, general resources, and class options.
- ▶ Retrieve RACF user password and password phrase envelopes.

The SDBM back end uses the R_admin “run command” interface to implement portions of the **adduser**, **addgroup**, **rdefine**, **altuser**, **altgroup**, **ralter**, **permit**, **setropts**, **deluser**, **delgroup**, **rdelete**, **connect**, **remove**, and **search** RACF commands. However, the SDBM back end uses the R_admin profile extract functions to retrieve specific user, group, connection, and resource information about search and compare requests. Also, the SDBM back end uses the **R_admin setropts** extract function to retrieve class options information when a search or compare request is performed on the SETROPTS entry. The keywords of each of these RACF commands are mapped to LDAP attribute types to enable adding, modifying, and searching RACF data. See the Accessing RACF information chapter in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for the mappings between the RACF command keywords and LDAP attribute types. These attribute types are already part of the initial or minimum schema, and therefore it is not necessary to modify the schema to add them or to use the SDBM back end.

Advantages

- ▶ Ability to remotely manage RACF users, groups, user-group connections, and general resource profiles over the LDAP protocol. Thus, the RACF administrator does not need to be logged directly into a 3270 session to issue RACF commands. The RACF commands issued run under the authority of the authenticated or bound RACF user.
- ▶ Ability to add, modify, and display user and group custom fields on RACF users and groups.

Considerations

- ▶ Hardcoded directory hierarchy
- ▶ There are limited number of searches and search filters that are supported in the SDBM back end. See 3.5.3, “Searching the SDBM back end” on page 52 for more information.
- ▶ Only one SDBM back end can be configured.
- ▶ Only RACF data (users, groups, user-group connections, and general resource profiles) can be stored or accessed through the SDBM back end.

3.5.1 SDBM Configuration

IBM Tivoli Directory Server for z/OS can be configured with an SDBM back end by either using the **dsconfig** utility to automate the process, or configuration can be performed manually for situations where the **dsconfig** utility is not adequate or further configuration updates are required.

Automated Configuration

The LDAP configuration utility, **dsconfig**, can simplify and automate the LDAP server configuration process for the SDBM back end. There are no members generated specifically

for SDBM. However, there are general use members that are generated and help to deploy an LDAP server with SDBM.

The **dsconfig** utility produces a number of files, including the DSCONFIG and DSENVVAR LDAP server configuration and environment variables files, and a procedure member needed to start the LDAP server as a started task. Using these files, the LDAP server can be started with an SDBM back end.

To configure the SDBM back end using the **dsconfig** utility, specify a value for the SDBM_SUFFIX parameter in the ds.profile file and this creates an SDBM back end database section in the generated DSCONFIG file. If RACF general resource profiles are to be managed by the SDBM back end (only available in z/OS V1R11 or later), set the SDBM_ENABLELERESOURCES parameter in ds.slapped.profile to ON.

For a complete step by step process of using the **dsconfig** utility, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

Manual Configuration

The SDBM back end can be configured manually if the **dsconfig** utility is not adequate or if further configuration updates are required. A roadmap for configuring an SDBM back end can be found in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

The SDBM back end-specific section of the ds.conf configuration file contains configuration options that are specific to the SDBM back end. To configure the LDAP server to run with an SDBM back end, the configuration files can first be copied from the /usr/lpp/ldap/etc directory. Figure 3-8 is a sample SDBM back end-specific section of the ds.conf configuration file.

```
database sdbm GLDBSD31/GLDBSD64
suffix cn=racf
enableResources on
```

Figure 3-8 SDBM back end section

Note: Although the suffix in this example is cn=racf, this value can be any valid DN suffix.

The database and suffix configuration options are required for configuration of an SDBM back end.

The following options are optional for configuring an SDBM back end: **enableResources**, **include**, **readOnly**, **sizeLimit** and **timeLimit**.

In z/OS V1R11, the ability to remotely manage general resource profiles (however not RACF dataset profiles) and modify SETROPTS settings that directly affect classes was added to the SDBM back end. Thus, the **enableResources** configuration option was introduced in z/OS V1R11 to add those portions of the SDBM directory hierarchy. If this option is not specified or set to off, this feature is not provided.

For more details regarding configuration options, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

SDBM back end directory hierarchy

When the SDBM back end is configured, the top level entries are pseudo entries that help to better organize the data that is present in the RACF database. The users, groups, user-group, and each RACF resource class exists as a pseudo entry directly underneath the SDBM suffix.

Figure 3-9 on page 49 displays the directory hierarchy when **enableResources** off is specified in the SDBM back end section.

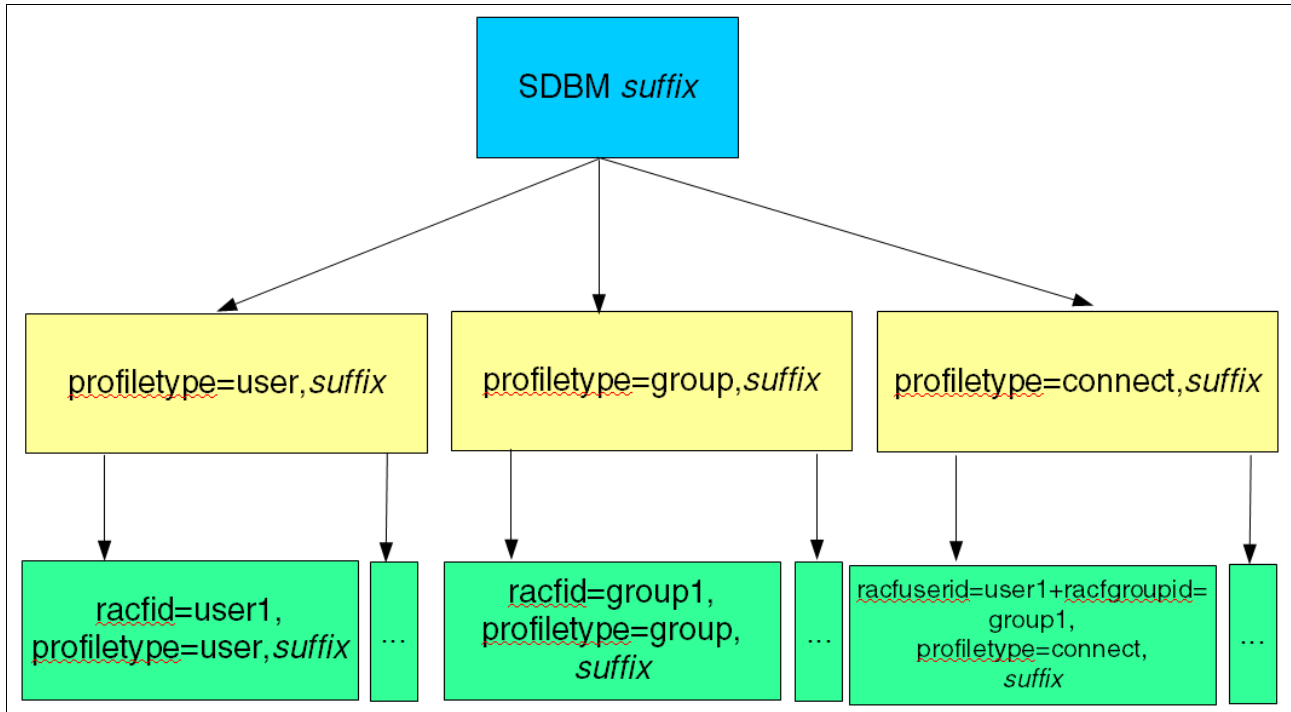


Figure 3-9 SDBM directory hierarchy for users, groups, and connect

If **enableResources on** is specified in the SDBM back end section of the configuration file, Figure 3-10 displays the additions to the SDBM directory hierarchy. Each active RACF general resource class exists as a pseudo entry and under those entries is each RACF general resource profile.

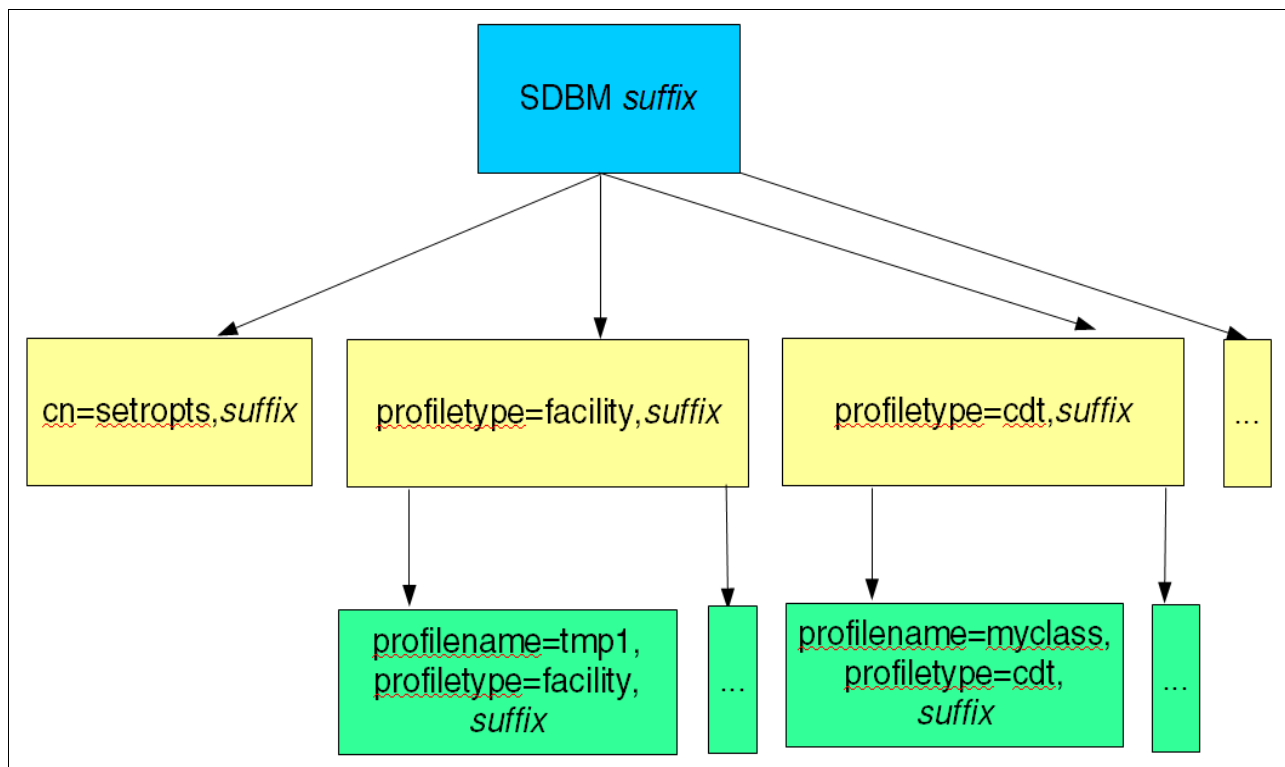


Figure 3-10 SDBM directory hierarchy for setropts and resource profiles

3.5.2 Using the SDBM back end

SDBM back end operations can be performed after several types of binds to the LDAP server. In each of these binds, the LDAP server associates a RACF user ID with the bound user. SDBM invokes RACF commands under the context of this RACF user ID, and RACF uses its normal authorization processing to determine what this RACF user ID can do.

The supported bind mechanisms are:

- ▶ Simple bind to SDBM: The RACF user ID is specified in the bind DN.
- ▶ LDBM, TDBM, or CDBM native authentication bind: The RACF user ID specified in the native authentication entry is used.
- ▶ GSSAPI (Kerberos) bind: The RACF user ID is mapped by SDBM from the Kerberos identity.
- ▶ EXTERNAL (SSL) bind: The RACF user ID associated with the certificate is used if certificate mapping is successfully performed.

See 5.2, “Authentication mechanisms supported by IBM Tivoli Directory Server for z/OS” on page 92 for additional information about these bind mechanisms.

Adding a WORKATTR segment

The following is an example of modifying existing RACF user, u1234, to add a WORKATTR segment with values specified for the WABLDG and WADEPT keywords, with the z/OS **1dapmodify** utility.

1. Create file `modu1234.1dif` that has the contents shown in Example 3-3 on page 51.

Example 3-3 *ldif contents*

```
dn: racfid=u1234,profiletype=user,cn=racf
changetype: modify
add: racfBuilding
racfBuilding: 256
-
add: racfDepartment
racfDepartment: LDAP
```

2. Execute the **ldapmodify** utility to modify the RACF u1234 user ID:

```
ldapmodify -D "racfid=radmin,profiletype=user,cn=racf" -w radminpw -f
modu1234.ldif
```

3. Modify the RACF user in the SDBM back end using the following RACF ALTUSER command, which runs the authority of the bound, radmin, RACF user.

```
ALTUSER U1234 WORKATTR(WABLDG('256') WADEPT('LDAP'))
```

Displaying a user-group connection using **ldapsearch**

The following is an example of using the z/OS **ldapsearch** utility to display user-group connection, "racfuserid=u1234+racfgroupid=group1,profiletype=connect,cn=racf":

```
ldapsearch -L -D "racfid=radmin,profiletype=user,cn=racf" -w radminpw -b
"racfuserid=u1234+racfgroupid=group1,profiletype=connect,cn=racf" "objectclass=*"
```

The results of this command are shown in Example 3-4 on page 51.

Example 3-4 *SDBM connect entry results*

```
dn: racfuserid=U1234+racfgroupid=GROUP1,profiletype=CONNECT,cn=racf
racfuserid: U1234
racfgroupid: GROUP1
racfconnectauthdate: 02/08/10
racfconnectowner: RACFID=RADMIN,PROFILETYPE=USER,CN=SDBM
racfconnectgroupauthority: USE
racfconnectgroupuacc: NONE
racfconnectcount: 0
objectclass: TOP
objectclass: RACFBASECOMMON
objectclass: RACFCONNECT
```

The SDBM back end under the authority of the radmin RACF user does a LISTUSER U1234 and returns group information for GROUP1.

Creating a new resource profile using **ldapadd**

The following is an example of using the z/OS **ldapadd** utility to add a new resource profile for the TERMINAL class.

1. Create a file `term1.ldif` that has the contents shown in Figure 3-11.

```
dn: profilename=TERM1,profiletype=TERMINAL,cn=racf
objectclass: racfresource
racfOwner: GROUP1
racfUacc: NONE
racfaccesscontrol: ID(U2) ACCESS(READ)
```

Figure 3-11 *ldif contents*

2. Issue the following command:

```
ldapadd -D "racfid=admin,profiletype=user,cn=racf" -w adminpw -f term1.ldif
```

The SDBM back end under the authority of the admin RACF user does the following RACF commands:

```
RDEFINE TERMINAL TERM1 OWNER(GROUP1) UACC(NONE)
PERMIT TERM1 CLASS(TERMINAL) ID(U2) ACCESS(READ)
```

The RDEFINE is done first to create the new resource profile and then the PERMIT is done.

Refreshing a class with ldapmodify

The following is an example of using the z/OS ldapmodify utility to refresh the RACF FACILITY class.

1. Create file refresh.ldif that has the contents shown in Figure 3-12.

```
dn: cn=setropts,cn=racf
changetype: modify
replace: racfsetroptsattributes
racfsetroptsattributes: REFRESH
-
replace: racfraclist
racfraclist: profiletype=FACILITY,cn=racf
```

Figure 3-12 *ldif contents*

2. Issue the following command:

```
ldapmodify -D "racfid=admin,profiletype=user,cn=racf" -w admin -f
refresh.ldif
```

The SDBM back end under the authority of the admin RACF user executes the following RACF command:

```
SETOPTS REFRESH RACLIST(FACILITY)
```

3.5.3 Searching the SDBM back end

The SDBM back end supports a limited number of search filters based on the base distinguished name (DN) and scope of the search request. Most searches from one of the top level entries in the SDBM back end (e.g profiletype=user,suffix) only return the distinguished names of entries that match the search filter because they require a search of the entire RACF database. Generally to retrieve an entire or complete RACF user, group, user-group connection, general resource profile, or the setropts entry, a search request for the specific entry as the base distinguished name must be specified when performing the search.

Table 3-2 on page 53 illustrates the supported search filters and what is returned on different searches assuming that the authenticated user has the authority to perform the searches.

Note: The RACF general resource profile and setropts entries are only returned when the enableResources on option is specified in the SDBM back end of the LDAP server configuration file.

Table 3-2 search filters

Search filter	Allowed base DN	Description and entries returned
objectclass=*	Any entry in the SDBM back end	Match any user, group, connection, resource profile, and setropts entry Entries returned: <ul style="list-style-type: none"> ▶ DN-only entries if the search scope includes all users, groups, user-group connections, resource profiles, or setropts ▶ Complete entry if scope only includes one entry
krbprincipalname= <i>any_value</i>	<i>suffix</i> profiletype=user, <i>suffix</i>	Complete user entry where the KERBNAME value in the KERB segment is equal to <i>any_value</i>
profilename= <i>any_value</i>	<i>suffix</i> profiletype=className, <i>suffix</i>	Finds the RACF general resource profiles whose names match <i>any_value</i> (can contain wildcards) DN-only entries returned
racfgroupid= <i>any_value</i>	<i>suffix</i> profiletype=connect, <i>suffix</i>	Finds connection profiles for members of the RACF groups whose names match <i>any_value</i> (can contain wildcards) DN-only entries returned
racfid= <i>any_value</i>	<i>suffix</i> profiletype=user, <i>suffix</i> profiletype=group, <i>suffix</i>	Finds user and group profiles for the RACF users and groups whose names match <i>any_value</i> (can contain wildcards) DN-only entries returned
racflnoteshortname= <i>any_value</i>	<i>suffix</i> profiletype=user, <i>suffix</i>	Complete entry where the RACF user profile has an LNOTES SNAME value equal to <i>any_value</i>
racfndsusername= <i>any_value</i>	<i>suffix</i> profiletype=user, <i>suffix</i>	Complete entry where the RACF user profile has an NDS UNAME value equal to <i>any_value</i>

Search filter	Allowed base DN	Description and entries returned
<code>racfomvsgroupid=number</code>	<code>suffix</code> <code>profilename=group,suffix</code>	Complete entry where the RACF group profile for one of the RACF groups has an OMVS GID values equal to <i>number</i>
<code>racfomvsgroupid;allOMVSids=number</code>	<code>suffix</code> <code>profilename=group,suffix</code>	Finds group profiles for all the RACF groups whose OMVS GID values match <i>number</i> DN-only entries returned
<code>racfomvsuid=number</code>	<code>suffix</code> <code>profilename=user,suffix</code>	Finds user profiles for all the RACF users whose OMVS UID values match <i>number</i> DN-only entries returned
<code>racfomvsuid;</code> <code>allOMVSid=number</code>	<code>suffix</code> <code>profilename=user,suffix</code>	Finds user profiles for all the RACF users whose OMVS UID values match <i>number</i> DN-only entries returned
<code>racfuserid=any_value</code>	<code>suffix</code> <code>profilename=connect,suffix</code>	Finds connection profiles for RACF users whose names match <i>any_value</i> (can contain wildcards) DN-only entries returned
<code>(&(racfuserid=any_value1)</code> <code>(racfgroupid=any_value2))</code>	<code>suffix</code> <code>profilename=connect,suffix</code>	Finds connection profiles for RACF users whose names match <i>any_value1</i> and who belong to RACF groups whose names match <i>any_value2</i> (both can contain wildcards) DN-only entries returned

See the “Accessing RACF information” chapter in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for more information about the supported searches and search filters in the SDBM back end.

3.5.4 Tuning the SDBM back end (RACF database)

If users perform SDBM simple binds, native authentication binds, or SASL EXTERNAL binds where the certificate is mapped to a RACF user, the LAST-ACCESS date and time are updated by default in the RACF database. These logins can cause unanticipated updates to the RACF database for each user's login. RACF provides a manner in which only the first user login for the day is recorded in the LAST-ACCESS day and time, reducing the number of concurrent updates that are being made to the RACF database. To enable this feature, assuming that your LDAP server started task proc is GLDSRV, enter the following RACF commands:

```
RDEFINE APPL GLDSRV UACC(READ) APPLDATA('RACF-INITSTATS(DAILY)')
SETROPTS RACLIST(APPL) REFRESH
```

The SDBM back end uses the R_admin run-command and R_admin profile extract interfaces. Generally, the performance of the SDBM back end depends on how the RACF database is tuned. For information about tuning your RACF database to improve performance in general, see *z/OS V1R12.0 Security Server RACF System Programmer's Guide, SA22-7681*.

3.5.5 RACF resources

The SDBM back end makes RACF information available using the LDAP protocol, this includes access to RACF General Resources. Any existing General Resource classes can be targeted, along with defining and using classes in the CDT class.

Below is an example of how to use LDAP to define a new class in the CDT class, and then define a profile type within the new class.

The RACF commands will be shown and then the equivalent LDAP commands.

Define a new class, activate it and refresh the newly created class.

Using RACF commands this would be accomplished by a series of commands similar to Figure 3-13.

```
RDEFINE CDT TSTCLAS1 UACC(NONE) CDTINFO( DEFAULTUACC(NONE) FIRST(ALPHA)
MAXLENGTH(42) OTHER(ALPHA,NUMERIC,SPECIAL) POSIT(303) RACLIST(REQUIRED)
SECLABELSREQUIRED(YES) )

RALTER CDT TSTCLAS1 AUDIT(FAILURES(READ)) WARNING

SETROPTS RACLIST(CDT)
SETROPTS CLASSACT(CDT)
SETROPTS RACLIST(CDT) REFRESH
```

Figure 3-13 RACF commands to create new class

The equivalent steps using LDAP commands are:

1. Define the class:

```
ldapmodify -D racfid=admin,profiletype=user,cn=myracf -w secret -a -f tstclas1.add.ldif
```

tstclas1.add.ldif has the contents shown in Figure 3-14.

```
dn: profilename=tstclas1,profiletype=cdt,cn=myRacf
racfuacc: none
racfcdtinfoDefaultUacc: none
racfcdtinfoFirst: alpha
racfcdtinfoMaxLength: 0042
racfcdtinfoOther: alpha numeric special
racfcdtinfoPosit: 303
racfcdtinfoRaclist: required
racfcdtinfoSecLabelsRequired: yes
racfMemberList: planner
racfMemberList: manager
```

Figure 3-14 Ldif file to create RACF class

2. Alter the class to set audit parameters:

```
ldapmodify -D racfid=admin,profiletype=user,cn=myracf -w secret -f tstclas1.mod.ldif
```

tstclas1.mod.ldif has the contents shown in Figure 3-15 on page 56.

```
dn: profilename=TSTclas1,profiletype=cdt,cn=myRacf
changetype: modify
add:x
racfResourceAudit: failures(read)
racfResourceAttributes: warning
```

Figure 3-15 Idif file to alter audit parameters

3. Add the class to RACLIST:

```
ldapmodify -D racfid=admin,profiletype=user,cn=myracf -w secret -f raclist.ldif
```

raclist.ldif has the contents shown in Figure 3-16.

```
dn: cn=setropts,cn=myRacf
changetype: modify
add:x
racFRACLIST: cdt
```

Figure 3-16 Idif file to add class to RACLIST

4. Activate the class:

```
ldapmodify -D racfid=admin,profiletype=user,cn=myracf -w secret -f activate.ldif
```

activate.ldif has the contents shown in Figure 3-17.

```
dn: cn=setropts,cn=myRacf
changetype: modify
add:x
racClassAct: cdt
```

Figure 3-17 Idif file to activate class

5. Refresh the class:

```
ldapmodify -D racfid=admin,profiletype=user,cn=myracf -w secret -f refresh.ldif
```

refresh.ldif has the contents shown in Figure 3-18.

```
dn: cn=setropts,cn=myRacf
changetype: modify
add:x
racFRACLIST: cdt
RACFSETROPTSATTRIBUTES: REFRESH
```

Figure 3-18 Idif file to refresh RACLIST class

List and Search information about the new class

To list and search for information about the new class using RACF commands, use commands similar to those shown in Figure 3-19.

```
RLIST CDT * CDTINFO

CDT          TSTCLAS1
...
CDTINFO INFORMATION
CASE = UPPER
DEFAULTRC = 004
DEFAULTUACC = NONE
FIRST = ALPHA
GENLIST = DISALLOWED
GENERIC = ALLOWED
GROUP =
KEYQUALIFIERS = 0000000000
MACPROCESSING = NORMAL
MAXLENGTH = 042
MAXLENX = NONE
MEMBER =
OPERATIONS = NO
OTHER = ALPHA NUMERIC SPECIAL
POSIT = 0000000303
PROFILESALLOWED = YES
RACLIST = REQUIRED
SECLABELSREQUIRED = YES
SIGNAL = NO

SEARCH CLASS(CDT)

TSTCLAS1
```

Figure 3-19 List and search information about new class with output

The following example lists the class using LDAP commands:

```
ldapsearch -D racfid=admin,profiletype=user,cn=myracf -w secret -s base -b
PROFILENAME=TSTCLAS1,profiletype=CDT,cn=myracf objectclass=*
```

which would return output similar to Figure 3-20 on page 58.

```

PROFILENAME=TSTCLAS1,profiletype=CDT,cn=myracf
...
racfcdtinofoposit=303
racfcdtinofomaxlength=42
racfcdtinofodefaultrc=4
racfcdtinofkeyqualifiers
racfcdtinofirst=ALPHA
racfcdtinofooter=ALPHA
racfcdtinofooter=NUMERIC
racfcdtinofooter=SPECIAL
racfcdtinofoperations=NO
...
racfcdtinofsignal=NO
racfcdtinofocase=UPPER
racfcdtinofogeneric=ALLOW
objectclass=TOP
objectclass=RACFRESOURCE
objectclass=EXTENSIBLEOBJECT

```

Figure 3-20 *ldapsearch output*

The following searches the CDT for classes:

```
ldapsearch -D racfid=admin,profiletype=user,cn=myracf -w secret -s one -b
profiletype=CDT,cn=myracf profilename=*
```

which would return:

```

...
profilename=TSTCLAS1,profiletype=CDT,cn=myracf
...

```

The following is another way to perform the search:

```
ldapsearch -D racfid=admin,profiletype=user,cn=myracf -w secret -s one -b
cn=myracf objectclass=*
```

which would return

```

...
profiletype=TSTCLAS1,cn=myracf
...

```

Define a profile in a class and permit users and groups to the new profile

To define a new profile in the newly created class and to permit select users and groups to the new profile could be done with the following RACF commands shown in Figure 3-21 on page 59.

```

RDEFINE TSTCLAS1 TSTPROF1 APPLDATA('Resource profile TSTPROF1 in class
TSTCLAS1') UACC(NONE) WARNING

RALTER TSTCLAS1 TSTPROF1 AUDIT(SUCCESS(READ)) UACC(READ) NOWARNING

SETOPTS RACLIST(TSTCLAS1)
SETOPTS CLASSACT(TSTCLAS1)
SETOPTS RACLIST(TSTCLAS1) REFRESH

PERMIT TSTPROF1 ACCESS(UPDATE) CLASS(TSTCLAS1) ID(USER1 USER2 USER3 GRP1 GRP2
GRP3)

SETOPTS RACLIST(TSTCLAS1) REFRESH

```

Figure 3-21 Define a profile and permit users and groups

This is accomplished in LDAP using the following steps:

1. Define a new profile in the class:

```

ldapmodify -D racfid=admin,profiletype=user,cn=myRacf -w secret -a -f
tstprof1.add.ldif

```

tstprof1.add.ldif has the contents shown in Figure 3-22.

```

dn: profilename=TSTPROF1,profiletype=tstclas1,cn=myRacf
racfApp1Data: 'Resource profile TSTPROF1 in class TSTCLAS1'
racfUacc: none
racfResourceAttributes: warning

```

Figure 3-22 Define a new profile in a class

2. Alter the profile to set audit parameters:

```

ldapmodify -D racfid=admin,profiletype=user,cn=myRacf -w secret -f
tstprof1.mod.ldif

```

tstprof1.mod.ldif has the contents shown in Figure 3-23.

```

dn: profilename=TSTPROF1,profiletype=tstclas1,cn=myRacf
changetype: modify
add:x
racfResourceAudit: success(read)
racfUacc: read
racfResourceAttributes: nowarning

```

Figure 3-23 Alter audit settings

3. Refresh the class:

```

ldapmodify -D racfid=admin,profiletype=user,cn=myRacf -w secret -f
tstclas1.refresh.ldif

```

tstclas1.refresh.ldif has the contents shown in Figure 3-24 on page 60.

```
dn: cn=setropts,cn=myRacf
changetype: modify
add:x
racfRACLIST: TSTCLAS1
RACFSETOPTSATTRIBUTES: REFRESH
```

Figure 3-24 Refresh

4. Activate the class:

```
ldapmodify -D racfid=admin,profiletype=user,cn=myracf -w secret -f
tstclas1.activate.ldif
```

tstclas1.activate.ldif has the contents shown in Figure 3-25.

```
dn: cn=setropts,cn=myRacf
changetype: modify
add:x
racfClassAct: TSTCLAS1
```

Figure 3-25 Activate

5. Give users access to the profile:

```
ldapmodify -D racfid=admin,profiletype=user,cn=myracf -w secret -f permit.ldif
```

permit.ldif has the contents shown in Figure 3-26.

```
dn: profilename=TSTPROF1,profiletype=tstclas1,cn=myRacf
changetype: modify
add:x
RACFACCESSCONTROL: ID(USER1 GRP1 GRP2 GRP3) ACCESS(update)
racfaccesscontrol: id(user2) access(update)
racfaccesscontrol: id(user3) access(update)
```

Figure 3-26 Permit

6. Refresh the profile:

```
ldapmodify -D racfid=admin,profiletype=user,cn=myracf -w secret -f
tstclas1.refresh.ldif
```

tstclas1.refresh.ldif has the contents shown in Figure 3-27.

```
dn: cn=setropts,cn=myRacf
changetype: modify
add:x
racfRACLIST: TSTCLAS1
RACFSETOPTSATTRIBUTES: REFRESH
```

Figure 3-27 Refresh

List and Search the new profile

The following RACF command would be used to list the new profile:

```
RLIST TSTCLAS1 TSTPROF1 ALL
```

The command returns results similar to Figure 3-28.

```
TSTCLAS1  TSTPROF1
LEVEL  OWNER          UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----  -----
00     SUIMGJT          NONE              READ         NO
INSTALLATION DATA
-----
NONE
APPLICATION DATA
-----
RESOURCE PROFILE TSTPROF1 IN CLASS TSTCLAS1
SECLEVEL
-----
NO SECLEVEL
CATEGORIES
-----
NO CATEGORIES
SECLABEL
-----
NO SECLABEL
AUDITING
-----
SUCCESS(READ)
NOTIFY
-----
NO USER TO BE NOTIFIED
CREATION DATE  LAST REFERENCE DATE  LAST CHANGE DATE
(DAY) (YEAR)    (DAY) (YEAR)         (DAY) (YEAR)
-----
08
ALTER COUNT    CONTROL COUNT    UPDATE COUNT    READ COUNT
-----
000000         000000          000000         000000
USER          ACCESS  ACCESS COUNT
-----
USER1        UPDATE    000000
USER2        UPDATE    000000
USER3        UPDATE    000000
GRP1         UPDATE    000000
GRP2         UPDATE    000000
GRP3         UPDATE    000000
ID           ACCESS  ACCESS COUNT  CLASS          ENTITY  NAME
-----
NO ENTRIES IN CONDITIONAL ACCESS LIST
```

Figure 3-28 RLIST

The following RACF command would be used to search:

```
SEARCH CLASS(TSTCLAS1)
```

which would return:

```
TSTPROF1
```

This is accomplished using the following LDAP command:

```
ldapsearch -D racfid=admin,profiletype=user,cn=myracf -w secret -s base -b  
PROFILENAME=TSTprof1,profiletype=tstclas1,cn=myracf objectclass=*
```

which would return results similar to Figure 3-29.

```
PROFILENAME=TSTprof1,profiletype=tstclas1,cn=myracf  
profilename=TSTPROF1,profiletype=TSTCLAS1,cn=myracf  
profilename=TSTPROF1  
racfauthorizationdate=06/24/10  
racfowner=RACFID=admin,PROFILETYPE=USER,CN=MYRACF  
racflastreferencedate=06/24/10  
racflastchangedate=06/24/10  
racfalteraccesscount=0  
racfcontrolaccesscount=0  
racfupdateaccesscount=0  
racfreadaccesscount=0  
racfuacc=NONE  
racfresourceaudit=SUCCESS(READ)  
racflevel=0  
racfresourceglobalaudit=NONE  
racfaccesscontrol=ID(USER1) ACCESS(UPDATE) COUNT(0)  
racfAccessControl=ID(USER2) ACCESS(update) COUNT(0)  
racfAccessControl=ID(USER3) ACCESS(update) COUNT(0)  
racfAccessControl=ID(GRP1) ACCESS(update) COUNT(0)  
racfAccessControl=ID(GRP2) ACCESS(update) COUNT(0)  
racfAccessControl=ID(GRP3) ACCESS(update) COUNT(0)  
objectclass=TOP  
objectclass=RACFRESOURCE  
objectclass=EXTENSIBLEOBJECT
```

Figure 3-29 ldapsearch

Delete the new profile and the new class

The RACF commands shown in Figure 3-30 would be used to delete the new profile and class and verify their deletion.

```
RDELETE TSTCLAS1 TSTPROF1
SETROPTS RACLIST(TSTCLAS1) REFRESH
SETROPTS NOCLASSACT(TSTCLAS1)

RLIST TSTCLAS1 TSTPROF1 ALL
TSTPROF1 NOT FOUND

SEARCH CLASS(TSTCLAS1) FILTER(TSTPROF1)
NO ENTRIES MEET SEARCH CRITERIA

RDELETE CDT TSTCLAS1
SETROPTS RACLIST(CDT) REFRESH
SETROPTS NOCLASSACT(CDT)

RLIST CDT * CDTINFO
NOTHING TO LIST

SEARCH CLASS(CDT)
ENTRIES MEET SEARCH CRITERIA
```

Figure 3-30 Delete new profile and class

The following series of **ldap** commands would delete the new profile and class.

1. Delete the new profile:

```
ldapdelete -D racfid=admin,profiletype=user,cn=myracf -w secret
PROFILENAME=tstPROF1,profiletype=TSTCLAS1,cn=myracf
```

2. Refresh the class:

```
ldapmodify -D racfid=admin,profiletype=user,cn=myracf -w secret -f
tstclas1.refresh.ldif
```

The contents of `tstclas1.refresh.ldif` are shown in Figure 3-31.

```
dn: cn=setropts,cn=myRacf
changetype: modify
add:x
racfRACLIST: TSTCLAS1
RACFSETROPTSATTRIBUTES: REFRESH
```

Figure 3-31 Refresh

3. Deactivate the class:

```
ldapmodify -D racfid=admin,profiletype=user,cn=myracf -w secret
tstclas1.deact.ldif
```

The contents of `tstclas1.deact.ldif` are shown in Figure 3-32 on page 64.

```
where tstclas1.deact.ldif is:
dn: cn=setropts,cn=myRacf
changetype: modify
delete:racfclassact
racfClassAct: TSTCLAS1
```

Figure 3-32 Deactivate class

4. Verify that the profile has been deleted:

```
ldapsearch -D racfid=admin,profiletype=user,cn=myracf -w secret -s base -b
PROFILENAME=TSTprof1,profiletype=tstclas1,cn=myracf objectclass=*
```

The search command should return results similar to those shown in Figure 3-33.

```
ldap_search: No such object
ldap_search: matched: PROFILETYPE=TSTCLAS1,CN=MYRACF
ldap_search: additional info: R004071 DN
'PROFILENAME=TSTPROF1,PROFILETYPE=TSTCLAS1,CN=MYRACF' does not exist
(sdbm_search)
```

Figure 3-33 Search

The following search reveals no profiles defined for the class:

```
ldapsearch -D racfid=admin,profiletype=user,cn=myracf -w secret -s one -b
profiletype=TSTCLAS1,cn=myracf profilename=tstPROF1
```

This command should return:

```
no output
```

5. Delete the class from the CDT:

```
ldapdelete -D racfid=admin,profiletype=user,cn=myracf -w secret
PROFILENAME=tstclas1,profiletype=cdt,cn=myracf
```

6. Refresh the CDT:

```
ldapmodify -D racfid=admin,profiletype=user,cn=myracf -w secret -f refresh.ldif
```

The contents of refresh.ldif are shown in Figure 3-34.

```
dn: cn=setropts,cn=myRacf
changetype: modify
add:x
racfraclist: cdt
RACFSETOPTSATTRIBUTES: REFRESH
```

Figure 3-34 Refresh

3.6 GDBM back end

The z/OS LDAP server provides the GDBM back end to store change logging information. The GDBM back end can use either a DB2 or a z/OS UNIX System Services file system for its data storage facility.

The change log consists of a set of entries in the GDBM directory that contain information about changes to TDBM, LDBM, or CDBM entries, the LDAP server schema entry (cn=schema), or to an object controlled by an application (such as a RACF user, group, group membership, or general resource profile).

Each LDAP server can contain only one change log. The change log entries are created in the same order as the changes are made, and each change log entry is identified by a change number value, beginning with 1. The change number value is incremented each time a change number is assigned to a change log entry. In this manner, the change number of a new change log entry is always greater than all change numbers of existing change log entries.

Change log entries are only created by the LDAP server. The user cannot directly add a change log entry. The user also cannot modify or rename a change log entry. However, the user can delete change log entries and manually trim the change log.

TDBM, LDBM, CDBM, and schema changes

When change logging is activated, each add, modify, delete, or modify DN operation of an entry from a TDBM, LDBM, or CDBM back end, or each modify of the LDAP server schema entry, results in the creation of a change log entry. Change log entries are not created for entries that are added to a TDBM back end when using the **1d1f2ds** load utility.

RACF changes

RACF can be customized to create LDAP change log entries when a change is made to a user, group, group membership, or general resource profile.

The **changeLogAddEntry** extended operation allows an application to log changes to data that it controls. This interface is used by RACF to log adds, modifies, and deletes of a RACF user, group, group membership, or general resource profile. The RACF changes can either be driven through the LDAP server through the SDBM back end or directly to RACF.

Change log entries

The change log directory consists of the following:

- ▶ A root (suffix) entry, named cn=changelog
- ▶ One or more leaf entries, named changenumber=nnn,cn=changelog

Each change log entry is created by the LDAP server as a leaf entry directly under the change log root entry, using attributes from the **changeLogEntry** and **ibm-changeLog** objectclasses.

Change log entries are deleted by the LDAP server when the change log is trimmed because of reaching a limit specified by the **changeLogMaxEntries** or **changeLogMaxAge** options in the configuration file. Change log entries can also be deleted by the user through a normal delete operation.

Change log schema

The following object classes and attributes define a change log entry. These object classes and attributes are defined by default in the LDAP server schema.

- ▶ objectclass: **changeLogEntry**
 - **changenumber**: An integer assigned to the change log entry
 - **targetDN**: The DN to which the change was applied. For RACF, this DN is created from a user, group, class, or resource name passed in by RACF and the SDBM suffix

- **changeType**: Can select add | modify | delete | modrdn
- **changeTime**: The time stamp of when the change was made (not when this entry was created)
- **changes**: The added entry or the modifications, in LDIF format
- **newRDN**: The new RDN specified in a TDBM, LDBM, or CDBM modify DN operation
- **deleteOldRdn**: A boolean indicating if the old RDN was deleted in a TDBM, LDBM, or CDBM modify DN operation
- **newSuperior**: The new superior distinguished name specified in a TDBM, LDBM, or CDBM modify DN operation
- ▶ objectclass: **ibm-changelog**
 - **ibm-changelogInitiatorsName**: The DN of the entity that initiated the change. For RACF, this DN is created from a user ID passed in by RACF and the SDBM suffix.

The change log root entry and change log entries also have the standard operational attributes: the ACL attributes, creatorsname, createtimestamp, modifiersname, modifytimestamp, and ibm-entryuuid (change log root only).

Advantages

- ▶ GDBM provides change logging.
- ▶ Entries can be stored in DB2 or in a UNIX System Services file system.
- ▶ The file-based GDBM back end runs in both 31-bit and 64-bit modes.
- ▶ Change log entries can be searched and deleted by a user application.

Considerations

- ▶ The LDAP server must be started in 31-bit mode if using DB2-based GDBM.

3.6.1 GDBM configuration

The LDAP server can be configured with a GDBM back end by either using the **dsconfig** utility to automate the process, or configuration can be performed manually for situations where the **dsconfig** utility is not adequate or further configuration updates are required.

Automated configuration

The LDAP configuration utility, **dsconfig**, can simplify and automate the LDAP server configuration process for GDBM (among other back ends).

For DB2-based GDBM:

For DB2-based GDBM configuration specifically, **dsconfig** will generate the following members:

- ▶ DBCLI: A JCL job that binds the CLI packages to DB2 and the DSNACLI plan.
- ▶ DSNAOINI: The DB2 CLI initialization file.
- ▶ GDBSPUFI member: A set of DB2 SQL statements, to be executed using the SPUFI tool, that defines database tables for the GDBM DB2-based back end.

After DB2 has been started, the DBCLI JCL can be submitted to bind the CLI packages to DB2 and the DSNACLI plan. The GDBSPUFI member can then be submitted through the DB2 SPUFI interactive tool to define database tables for the GDBM back end.

For file-based GDBM:

There are no members generated specifically for a file-based GDBM.

The **dsconfig** utility produces a number of other files, including the DSCONFIG and DSENVVAR LDAP server configuration and environment variables files, and a procedure member needed to start the LDAP server as a started task. Using these files (along with the DSNAOINI CLI initialization file if configuring DB2-based GDBM) the LDAP server can be started with a GDBM back end.

For a complete step by step process of using the **dsconfig** utility, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

Manual configuration

The GDBM back end can be configured manually if the **dsconfig** utility is not adequate or if further configuration updates are required. A roadmap for configuring a GDBM back end can be found in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05. After DB2 or a z/OS UNIX System Services file system has been set up for GDBM, the LDAP server is ready to be configured and started.

The DB2-based and file-based GDBM back end-specific sections of the `ds.conf` configuration file contain configuration options that are specific to the GDBM back end. Only one GDBM back end can be configured, either DB2-based or file-based. To configure the LDAP server to run with a GDBM back end, the configuration files can first be copied from the `/usr/lpp/lap/etc` directory.

For DB2-based GDBM

The following is a sample DB2-based GDBM back end-specific section of the `ds.conf` configuration file:

```
# DB2-based GDBM database definition and configuration options
database GDBM GLDBGD31
dbuserid GLDSRV
```

GLDSRV is the DB2-based GDBM database owner.

The database and dbuserid configuration options are required for configuration of a DB2-based GDBM back end.

The following options are optional for configuring a DB2-based GDBM back end: **ac1SourceCacheSize**, **attrOverflowSize**, **changeLogging**, **changeLoggingParticipant**, **changeLogMaxAge**, **changeLogMaxEntries**, **dnToEidCacheSize**, **dsnaoini**, **entryCacheSize**, **entryOwnerCacheSize**, **filterCacheBypassLimit**, **filterCacheSize**, **include**, **persistentSearch**, **readonly**, **serverName**, **sizeLimit**, **timeLimit**.

For file-based GDBM

Following is a sample file-based GDBM back end-specific section of the `ds.conf` configuration file:

```
# File-based GDBM database definition and configuration options
database GDBM GLDBGD31/GLDBGD64
```

The database configuration option is required for configuration of a file-based GDBM.

The following options are optional for configuring a file-based GDBM back end: **changeLogging**, **changeLoggingParticipant**, **changeLogMaxAge**, **changeLogMaxEntries**, **commitCheckpointEntries**, **commitCheckpointTOD**, **databaseDirectory**, **fileTerminate**, **filterCacheBypassLimit**, **filterCacheSize**, **include**, **persistentSearch**, **readOnly**, **sizeLimit**, **timeLimit**.

For more details regarding configuration options, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

The attributes and object classes used by GDBM are always in the LDAP server schema.

3.6.2 Enabling change logging

Change logging is enabled by specifying a combination of configuration options in the GDBM, LDBM, TDBM, and CDBM back end configuration sections. The following options control change logging:

- ▶ **changeLogging**: This GDBM configuration option turns change logging on or off. The default, if not specified, is for change logging to be turned on.
- ▶ **changeLogMaxEntries** and **changeLogMaxAge**: These GDBM configuration options determine when removal of old change log entries takes place, otherwise known as trimming the change log. The default, if not specified, is not to trim the change log. If a change log entry exceeds the age limit set using the **changeLogMaxAge** configuration option, it is removed from the log. If the number of change log entries exceeds the limit set using the **changeLogMaxEntries** configuration option, the change log entries with the lowest changenumber values are removed.
- ▶ **changeLoggingParticipant**: This configuration option is used in the LDBM, TDBM, CDBM, or GDBM back end configuration sections. If specified in an LDBM, TDBM, or CDBM back end section, the option turns on change logging for changes to entries in that back end. If specified in the GDBM back end section, change logging is turned on for changes to the LDAP server schema. The default, if not specified, is to turn on change logging for TDBM, LDBM, and CDBM back ends and the LDAP server schema.

3.6.3 Additional configuration for RACF change logging

Additional configuration is required for RACF to be able to log changes to a RACF user, group, group membership, or resource profile:

- ▶ The SDBM back end must be configured. The SDBM suffix is needed to create a DN for the change log entry for a modification to a RACF user, group, group membership, or resource profile. SDBM is also required for retrieving the RACF user's new password or other changed fields. The following option must be specified in the SDBM section of the configuration file to allow change log entries to be created for changes to resource profiles:

```
enableResources on
```

- ▶ LDAP Program Call support must be enabled in the LDAP server containing the change log. To do this, add the following option to either the global section of the configuration file or to the command used to start the LDAP server:

```
listen ldap://:pc
```

The listen parameter for LDAP Program Call support is specified in addition to any other listen parameters.

RACF uses the RACFEVNT resource class to control the event types for which change log entries are created. If RACFEVNT is active, and the appropriate resource is protected by either a discrete or generic profile, LDAP change log entries are created for the corresponding event types on a system-wide basis.

Table 3-3 shows the name of the RACF resource in the RACFEVNT class that is used to control notifications for each type of supported change event.

Table 3-3 Resource in RACEVNT class

Resource in the RACEVNT class	Change event type
NOTIFY.LDAP.USER	<ul style="list-style-type: none"> ▶ Password and password phrase changes, regardless of the command or method used ▶ Updates to a user's revoke status (that is, changes to the FLAG4 field in the USER profile), regardless of the command or method used ▶ Users added using the ADDUSER command ▶ User modifications made using the ALTUSER or PASSWORD command ▶ Users deleted using the DELUSER command
NOTIFY.LDAP.GROUP	<ul style="list-style-type: none"> ▶ Groups added using the ADDGROUP command ▶ Group modifications made using ALTGROUP command ▶ Groups deleted using the DELGROUP command
NOTIFY.LDAP.CONNECT	<ul style="list-style-type: none"> ▶ Group membership changes made using any of the following commands: ALTUSER command, only when issued with GROUP, UACC, or AUTHORITY operand CONNECT command REMOVE command ▶ Users established in their default groups using the ADDUSER command
NOTIFY.LDAP.class-name	<ul style="list-style-type: none"> ▶ General resources added using the RDEFINE command ▶ General resource modifications made using the RALTER command ▶ Changes made using the PERMIT command to the standard or conditional access list of a general resource ▶ General resource deletions made using the RDELETE command

For more details about RACF change logging, see the *Security Server RACF Security Administrator's Guide*, SA22-7683.

Change log information in the root DSE entry

The following root DSE attributes enable applications to determine the location of the change log and the range of change numbers. The attributes appear whenever change logging is enabled (the GDBM back end is configured), whether or not change logging is currently on.

The location of the change log:

`changetlog=cn=changetlog`

The lowest change number currently in use in the change log. A zero indicates no change log entries:

`firstchangenumber=nnn`

The highest change number currently in use in the change log. A zero indicates no change log entries:

`lastchangenumber=nnn`

Unloading and loading the change log

The `ds21dif unload` utility cannot be used to unload the contents of the change log. The `search` operation should instead be used to unload GDBM. Change log entries cannot be loaded into the change log. Both the `add` operation and the `ldif2ds load` utility fail when processing change log entries.

3.6.4 Using the GDBM back end

The following example searches the root DSE for useful change log information:

- ▶ Invoke the `ldapsearch` utility to search the LDAP root DSE entry for change log information, such as the change log suffix and the lowest and highest change numbers currently in use. Example 3-5 shows partial output.

Example 3-5 ldapsearch example for changelog information

```
ldapsearch -D cn=admin -w secret -s base -b "" objectclass=*
changelog=cn=changelog
firstchangenumber=1
lastchangenumber=4
```

The following example searches for all entries in the GDBM back end:

- ▶ Invoke the `ldapsearch` utility to search the change log for all changes as shown in Example 3-6.

Example 3-6 ldapsearch example for all changes

```
ldapsearch -D cn=admin -w secret -b cn=changelog objectclass=*

changeNumber=1,cn=changelog
objectclass=top
objectclass=changeLogEntry
objectclass=ibm-changeLog
changenumber=1
changetype=add
targetdn=dc=yourcompany,dc=com
changes=objectClass: organization
objectClass: dcObject
description: The company domain component
o: Your Co.

ibm-changeinitiatorsname=cn=Admin
changetime=20100713205013.673322Z

changeNumber=2,cn=changelog
objectclass=top
objectclass=changeLogEntry
objectclass=ibm-changeLog
changenumber=2
changetype=add
targetdn=ou=users,dc=yourcompany,dc=com
changes=objectclass: top
objectclass: organizationalUnit
ou: users
description: Container for user entries
```

```
ibm-changeinitiatorsname=cn=Admin  
changetime=20100713205013.701144Z
```

```
changeNumber=3,cn=changelog  
objectclass=top  
objectclass=changeLogEntry  
objectclass=ibm-changeLog  
changenumber=3  
changetype=add  
targetdn=cn=Bill Barker,ou=users,dc=yourcompany,dc=com  
changes=objectClass: organizationalPerson  
sn: BaRkEr  
cn: Bill Barker  
telephoneNumber: 523-9494  
userPassword: *ComeAndGetIt*
```

```
ibm-changeinitiatorsname=cn=Admin  
changetime=20100713205013.718894Z
```

```
changeNumber=4,cn=changelog  
objectclass=top  
objectclass=changeLogEntry  
objectclass=ibm-changeLog  
changenumber=4  
changetype=add  
targetdn=cn=Joe Shmoe,ou=users,dc=yourcompany,dc=com  
changes=objectClass: organizationalPerson  
sn: Shmoe  
cn: Joe Shmoe  
telephoneNumber: 593-7777  
userPassword: *ComeAndGetIt*
```

```
ibm-changeinitiatorsname=cn=Admin  
changetime=20100713205013.736239Z
```

```
cn=changelog  
objectclass=top  
objectclass=container  
cn=changelog
```

3.6.5 Tuning the GDBM back end

The amount of space required to store a GDBM back end in a z/OS UNIX System Services file system depends on two things:

1. The maximum number of change log entries expected to be stored
2. The estimated size of a change log entry

The **changeLogMaxEntries** and **changeLogMaxAge** configuration options can be used to control the number of change log entries.

The size of a change log entry can be calculated based on the size of the LDIF used to add or modify a TDBM, LDBM, or CDBM entry because an entry's LDIF is inserted into the change

log entry. Therefore, the size of the largest entry's LDIF is used to estimate a change log entry's size.

The estimated space required to hold the GDBM back end data can be calculated as follows (this includes the additional space required to copy the database files during GDBM commit processing):

$6 \times (\text{maximum number of GDBM entries}) \times (\text{largest add or modify LDIF} + 1000)$



Schemas

This chapter discusses the IBM Tivoli Directory Server Schema.

4.1 Schema

Entries in the directory are made up of attributes that consist of an attribute type and one or more attribute values. These are referred to as attribute=value pairs. Every entry contains one or more objectclass=value pairs that identify what type of information the entry contains. The object classes associated with the entry determine the set of attributes that must or might be present in the entry.

The schema defines a set of attribute types that give characteristics to entries and further define an entry in the LDAP server other than just having a distinguished name (DN). For example, when describing a person you know, one could mention a person's eye color, hair color, height, weight, home address, email address, home telephone number, cell phone number, and whether he or she works or not. In an LDAP entry, each of these characteristics could be an attribute type and have one or more values.

The schema also defines object classes that are used to group attribute types together. An object class can then be used to define classifications of entries in the directory. For example, the statistics kept for players on a baseball team varies. For a baseball pitcher, the following types of statistics are kept: number of wins, saves, runs given up, walks allowed, strikeouts, and current/past teams. For a position player, such as an outfielder, the following types of statistics are kept: number of total hits, doubles, triples, home runs, runs batted in, stolen bases, positions played, and current/past teams. If you were to represent baseball players in an LDAP server, two object classes would be required because different types of statistics are kept for each player.

4.2 Schema configuration in IBM Tivoli Directory Server for z/OS

IBM Tivoli Directory Server for z/OS has a single schema for the entire server that lays out the foundation for the type of data stored in the directory. It defines the attribute types and the object classes that entries in the directory are allowed to have. IBM Tivoli Directory Server for z/OS stores its schema in an entry with a distinguished name (DN) of cn=schema.

In IBM Tivoli Directory Server for z/OS, the schema is stored in an z/OS UNIX System Services files system directory specified by the schemaPath configuration option in the LDAP server configuration file. The default for the schemaPath configuration option is the /var/ldap/schema directory.

If there is not already a schema present in the directory pointed to by the schemaPath configuration option, IBM Tivoli Directory Server for z/OS populates it with the minimum or initial schema. The minimum or initial schema can be used with the SDBM (without RACF custom fields), CDBM (with configuration related entries), and GDBM back ends. However, it needs to be updated for LDBM, TDBM, SDBM with RACF custom fields, and CDBM with user-defined entries.

Generally for each new IBM Tivoli Directory Server for z/OS release, there are new attribute types and object class definitions added to the minimum or initial schema. If migrating an existing IBM Tivoli Directory Server for z/OS to a more recent version of IBM Tivoli Directory Server for z/OS, these new attribute types and object class definitions are automatically added to the schema. This enables the server to automatically take advantage of any new features that require the new attribute types or object classes.

4.2.1 Applying schema to IBM Tivoli Directory Server for z/OS

If you are planning on using LDBM, TDBM, SDBM with RACF custom fields, or CDBM with user-defined entries, the minimum or initial schema is not sufficient for these needs. In these situations, it will be necessary to apply additional schema to IBM Tivoli Directory Server for z/OS.

IBM Tivoli Directory Server for z/OS ships two schema files, `schema.user.ldif` and `schema.IBM.ldif`, which are shipped in the `/usr/lpp/ldap/etc/` directory. The `schema.user.ldif` file contains schema from industry-standard RFCs and other products. The `schema.IBM.ldif` file contains schema that might be required for use with other IBM products.

Depending on the type of entries that you are planning on storing in these back ends, the schema definitions in `schema.user.ldif` and `schema.IBM.ldif` might be sufficient. If the schema definitions are sufficient, you can update the `cn=schema` entry by running the `ldapmodify` utility:

```
ldapmodify -D adminDN -w passwd -f /usr/lpp/ldap/etc/schema.user.ldif
ldapmodify -D adminDN -w passwd -f /usr/lpp/ldap/etc/schema.IBM.ldif
```

`adminDN` is the **adminDN** specified in the LDAP server configuration file and `passwd` is the password for the LDAP administrator's entry.

If the schema definitions in `schema.user.ldif` and `schema.IBM.ldif` are not sufficient for your directory needs, you might need to create your own schema definitions or obtain schema definitions from a third party if you are running an application that requires access to IBM Tivoli Directory Server for z/OS. See *Defining additional schema in z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for more information.

4.3 Attribute Types

Each attribute type that is defined in the schema has a data type, which is commonly referred to as the attribute syntax. The attribute syntax specifies the type of data that is allowed in its values when used within an entry. When adding or modifying entries in the directory, the schema enforces the attribute syntax to ensure that only allowed values can be added to entries in the directory. For example, the baseball player's statistics such as the number of total hits and doubles would be integers in the schema and the current/past teams would be character strings.

Table 4-1 shows the attribute syntaxes that are supported by IBM Tivoli Directory Server for z/OS. The schema also refers to each attribute syntax by a numeric OID (Object Identifier). These OIDs are defined in RFC 2252:

<http://www.ietf.org/rfc/rfc2252.txt>

In Table 4-1, the syntaxes marked with an * are only supported when the server compatibility (serverCompatLevel configuration option) is 6 or later. The server compatibility level of z/OS V1R10 is 4, z/OS V1R11 is 5, and z/OS V1R12 is 6.

Table 4-1 Supported attribute syntax

Attribute syntax	Attribute OID	Attribute description
Binary	1.3.6.1.4.1.1466.115.121.1.5	Binary data such as a jpeg photo

Attribute syntax	Attribute OID	Attribute description
Bit string*	1.3.6.1.4.1.1466.115.121.1.6	Binary bit data such as b'0110' or B'0110'
Boolean	1.3.6.1.4.1.1466.115.121.1.7	A true or false value
Certificate*	1.3.6.1.4.1.1466.115.121.1.8	Binary data (no format checking done)
Certificate List*	1.3.6.1.4.1.1466.115.121.1.9	Binary data (no format checking done)
Certificate Pair*	1.3.6.1.4.1.1466.115.121.1.10	Binary data (no format checking done)
Country String*	1.3.6.1.4.1.1466.115.121.1.11	Two character printable string (alphabetic, digits, ' (,), +,, -,., /,;,?, space)
Distinguished Name	1.3.6.1.4.1.1466.115.121.1.12	Sequence of attribute types-value pairs. Can be used to refer to other distinguished names in the directory.
Delivery Method*	1.3.6.1.4.1.1466.115.121.1.14	UTF-8 characters (no format checking)
Directory String	1.3.6.1.4.1.1466.115.121.1.15	UTF-8 characters
Enhanced Guide*	1.3.6.1.4.1.1466.115.121.1.21	UTF-8 characters (no format checking)
Facsimile Telephone Number*	1.3.6.1.4.1.1466.115.121.1.22	Printable string (alphabetic, digits, ' (,), +,, -,., /,;,?, space) and \$(no format checking)
Fax*	1.3.6.1.4.1.1466.115.121.1.23	Binary data (no format checking)
Generalized Time	1.3.6.1.4.1.1466.115.121.1.24	<p>yyymddhhmmss.fffff (local time)</p> <p>yyymddhhmmss.fffffZ (GMT)</p> <p>yyymddhhmmss.fffff-hhmm (Time zone west)</p> <p>yyymddhhmmss.fffff+hhmm (Time zone east)</p> <p>The seconds (ss) and microseconds (fffff) can be omitted and defaults to 0</p>
Guide*	1.3.6.1.4.1.1466.115.121.1.25	UTF-8 characters (no format checking)
IA5 String	1.3.6.1.4.1.1466.115.121.1.26	IA5 characters (commonly known as 7-bit ASCII)
Integer	1.3.6.1.4.1.1466.115.121.1.27	+/- 62 digit integer

Attribute syntax	Attribute OID	Attribute description
JPEG*	1.3.6.1.4.1.1466.115.121.1.28	Binary data (no format checking)
MHS OR Address*	1.3.6.1.4.1.1466.115.121.1.33	UTF-8 characters (no format checking)
Name And Optional UID*	1.3.6.1.4.1.1466.115.121.1.34	Sequence of attribute type and value pairs
Numeric String*	1.3.6.1.4.1.1466.115.121.1.36	List of space-separated numbers
Object Identifier	1.3.6.1.4.1.1466.115.121.1.38	Name or numeric object identifier
Other Mailbox*	1.3.6.1.4.1.1466.115.121.1.39	UTF-8 characters (no format checking)
Octet String	1.3.6.1.4.1.1466.115.121.1.40	Octet data
Postal Address*	1.3.6.1.4.1.1466.115.121.1.41	UTF-8 characters (no format checking)
Protocol Information*	1.3.6.1.4.1.1466.115.121.1.42	UTF-8 characters (no format checking)
Presentation Address*	1.3.6.1.4.1.1466.115.121.1.43	UTF-8 characters (no format checking)
Printable String*	1.3.6.1.4.1.1466.115.121.1.44	Printable string (alphabetic, digits, ' (,), +,,, -,., /;:,?, space)
Supported Algorithm*	1.3.6.1.4.1.1466.115.121.1.49	UTF-8 characters (no format checking)
Telephone Number	1.3.6.1.4.1.1466.115.121.1.50	Printable string (alphabetic, digits, ' (,), +,,, -,., /;:,?, and space) and "
Teletex Terminal Identifier*	1.3.6.1.4.1.1466.115.121.1.51	UTF-8 characters (no format checking)
Telex Number*	1.3.6.1.4.1.1466.115.121.1.52	Printable string (alphabetic, digits, ' (,), +,,, -,., /;:,?, space) and \$ (no format checking)
UTC Time	1.3.6.1.4.1.1466.115.121.1.53	Like Generalized Time above, but with a two-digit year specification (yy)

Matching rules are used by an LDAP server when performing search and compare operations on attribute types. These rules are also used to identify the value to be added or deleted when modifying entries, and when comparing a distinguished name with the name of an entry. There are three types of matching rules supported:

- ▶ EQUALITY is used to determine if two values are equal.
- ▶ ORDERING is used to determine how two values are ordered. This matching rule is used to determine what is returned when specifying a search filter such as (changeNumber>=5000) or (changeNumber<=5000).

- ▶ **SUBSTR** (substring) is used to determine if the presented value is a substring of an attribute value from the directory. This is used when wildcards, such as (cn=jo*), are present in a search filter. This search filter could return entries that have a cn attribute value equal to jon, joe, or john.

The matching rules along with the attribute syntax indicate the type of data that is allowed in an entry's attribute values. These checks are performed during add, modify, compare, and search operations involving attribute types. Each attribute syntax allows certain EQUALITY, ORDERING, and SUBSTR (substring) matching rules which indicates how each of the comparisons are performed. If the EQUALITY, ORDERING, or SUBSTR (substring) rules are not specified in the attribute type definition in the schema, there are defaults assigned and they are used. See the LDAP Directory Schema chapter in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for more information about the supported and default matching rules for each attribute syntax.

To illustrate how equality matching rules work, the cn=jon,o=ibm,c=us is in the directory as shown in Figure 4-1.

```
dn: cn=jon,o=ibm,c=us
objectclass: newPilotPerson
cn: jon
sn: cottrell
userpassword: secret
homephone: 555-555-5555
```

Figure 4-1 cn=jon,o=ibm,c=us sample entry

Note the following:

- The cn and sn attribute types have Directory string syntax and an EQUALITY matching rule of **caseIgnoreMatch**.
- The **userpassword** attribute type has Octet String syntax and an EQUALITY matching rule of **octetStringMatch**.
- The homephone attribute type has Telephone Number syntax and an EQUALITY matching rule of **telephoneNumberMatch**.

Equality matching rules tests with the cn=jon,o=ibm,c=us entry:

- Because the cn and sn attribute types have an EQUALITY matching rule of **caseIgnoreMatch**, the attribute value case that is specified on a search filter or a compare test does not matter; therefore (cn=jon) or (cn=JoN) are both equivalent and will successfully return the entry. However if the EQUALITY matching rule is **caseExactMatching**, only the (cn=jon) filter will successfully return the entry.
- Because the **userpassword** attribute type has an EQUALITY matching rule of **octetStringMatch**, a compare operation will only return the entry when using (userpassword=secret) as a test filter.

Note: IBM Tivoli Directory Server for z/OS does not support specifying attribute types that are Octet String syntax on a search filter.

- Because the homephone attribute type has an EQUALITY matching rule of **telephoneNumberMatch**, a compare or search operation will successfully return the entry when specifying either (homephone=555-555-5555) or (homephone=5555555555) on a search filter or a comparison test.

The ordering and substring matching rules work in a similar manner.

Attributes types defined in the schema can be designated as single-valued or multi-valued. A single-valued attribute is only allowed to have one value within an entry. A multi-valued attribute is allowed to have one or more values within an entry. There are times when multi-valued attributes are handy. For example, if devising an attribute type for the positions a baseball player can play, the position field ought to be a multi-valued attribute so that it can have multiple values in an entry. However, when describing the total number of hits a baseball player has in the current season, it should be defined as a single-valued attribute type.

Each attribute type in the schema has a usage that indicates if it is operational or non-operational. If the bound user is authorized to the attribute type, the value of an operational attribute type in an entry is only returned by a search operation if the attribute type is specified in the list of attribute to be returned or the plus '+' sign is specified. A non-operational attribute type is always returned assuming that the bound user is authorized to the attribute type.

It is also possible to define an attribute type in the schema so that users are restricted from being able to modify its values in an entry. These attribute types are called non-user modifiable attributes, and should only be specified for attribute types that are set by the server because they cannot be assigned a value by a user. The vast majority of attribute types in IBM Tivoli Directory Server for z/OS are user-modifiable.

IBM Tivoli Directory Server for z/OS supports classifying attribute types into access classes. The supported access classes are:

- ▶ Normal
- ▶ Sensitive
- ▶ Critical
- ▶ Restricted
- ▶ System

The access class assigned to an attribute type depends on the sensitivity of the data to be stored in the attribute type. For example, a baseball player's bank account number for direct depositing his paycheck ought to be considered a sensitive or critical attribute.

These attribute classes can then be used to grant users or groups authorization to an entire set of attribute types in the directory. The restricted and system attribute classes are used for attribute types such as `aclEntry` and `aclPropagate`, so those access classes should not be used when defining new attribute types. Generally, most attribute types in IBM Tivoli Directory Server for z/OS are in the normal access class. See 5.4, "Authorization using Tivoli Directory Server Access Control Lists (ACL)" on page 108 for more information.

4.3.1 **Attributetypes and ibmattributetypes attribute format**

The **attributetypes** and **ibmattributetypes** attribute values must be specified when adding a new attribute type to the `cn=schema` entry as shown in Figure 4-2 on page 80.

```
attributeTypes: ( numericoid [NAME qdescrs] [DESC qdstring] [OBSOLETE] [SUP oid]
[EQUALITY oid] [ORDERING oid] [SUBSTR oid] [SYNTAX noidlen] [SINGLE-VALUE]
[NO-USER-MODIFICATION] [USAGE attributeUsage] )
```

numericoid

The unique, assigned numeric object identifier.

NAME qdescrs

The name and alias names by which this attribute type is known. This is also known as the object identifier. The first name in the list is used as the base name and the other names are referred to as alias names.

DESC qdstring

Text description of the attribute type.

OBSOLETE

Indicates that the attribute type is obsolete.

SUP oid

Specifies the superior attribute type. When a superior attribute type is defined, the EQUALITY, ORDERING, SUBSTR, and SYNTAX values might be inherited from the superior attribute type. The referenced superior attribute type must also be defined in the schema. When the SYNTAX, EQUALITY, ORDERING, or SUBSTR values are not specified for an attribute type, the attribute type hierarchy is used to determine these values. The SYNTAX must be specified on the attribute type or through inheritance.

EQUALITY oid

Specifies the object identifier of the matching rule which is used to determine the equality of values.

ORDERING oid

Specifies the object identifier of the matching rule which is used to determine the order of values.

SUBSTR oid

Specifies the object identifier of the matching rule which is used to determine substring matches of values.

SYNTAX noidlen

The syntax defines the format of the data stored for this attribute type. It is specified using the numeric object identifier of the LDAP syntax and, optionally, the maximum length of data stored for this attribute type.

SINGLE-VALUE

Limits entries to only one value for this attribute type.

NO-USER-MODIFICATION

When specified, users might not modify values of this attribute type.

USAGE attributeUsage

Indicates whether the attribute type is operational or non-operational. An operational attribute is only returned when specifically requested on a search operation or the plus sign ('+') is specified. Specify userApplications for attributeUsage for a non-operational attribute or directoryOperation, distributedOperation, and DSAOperation for an operational attribute.

Figure 4-2 attributeTypes attribute format

The **ibmattributetypes** format is shown in Figure 4-3.

```
ibmattributetypes: ( numericoid [ACCESS-CLASS IBMAccessClass]
[RACFFIELD qdescrs] )

numericoid
The unique, assigned numeric object identifier of the associated attribute type.

ACCESS-CLASS ibmAccessClass
The level of sensitivity of the data values for this attribute type. The acceptable
values are normal, sensitive, and critical.

RACFFIELD qdescrs
The information needed to associate this attribute with a RACF custom field in a user or
group profile.

The format of qdescrs is either a value in single quotation marks:
RACFFIELD 'racFieldName'
or two values in parentheses, each in single quotation marks and separated by a blank:
RACFFIELD ('racFieldName' 'racFieldType')
```

racFieldName format must be: USER-CSDATA-name or GROUP-CSDATA-name

where name is the name of the associated RACF custom field. racFieldType is the type of custom field. The acceptable values are: char, flag, hex, num, and qchar. This value defaults to char if the racFieldType is not specified.

Figure 4-3 *ibmattributetypes* attribute format

See 4.6, “Defining additional schema for use with RACF custom fields” on page 87 for more information about modifying the schema for use with RACF user and group custom fields.

4.4 Object Classes

Object class definitions in the schema are used to group attribute types together to help define the characteristics of individual entries. The object class definition indicates the required and optional attribute types in an entry when it is created.

There are three types of supported object classes: structural, abstract, or auxiliary. When creating an entry in LDAP, only one structural object class can be specified. Abstract and auxiliary object classes are used to provide common characteristics to entries with different structural object classes. Abstract object classes are used to derive additional object classes. Abstract object classes must be referred to in a structural or auxiliary superior hierarchy. Auxiliary object classes are used to extend the set of required or optional attribute types of an entry. Therefore it is possible for an entry to include a structural object class and an auxiliary object class to add additional attribute types to the entry.

4.4.1 objectclasses attribute value format

The objectclasses attribute value must be specified when adding a new object class definition to the cn=schema entry as shown in Figure 4-4.

```
objectClasses: ( numericoid [NAME qdescrs] [DESC qdstring]
  [OBSOLETE] [SUP oids] [ABSTRACT|STRUCTURAL|AUXILIARY]
  [MUST oids] [MAY oids] )

numericoid
The unique, assigned numeric object identifier.

NAME qdescrs
The name and alias names by which this object class is known. This is also known as the
object identifier. The first name in the list is used as the base name. If name is not
specified, the numeric object identifier is used to refer to the object class.

DESC qdstring
Text description of the object class.

OBSOLETE
Indicates that the object class is obsolete.

SUP oids
List of one or more superior object classes. When a superior object class is defined,
entries specifying the object class must adhere to the superset of MUST and MAY values.
The supersets of MUST and MAY values include all MUST and MAY values specified in the
object class definition and all MUST and MAY values specified in the object class's
superior hierarchy. When an attribute type is specified as a MUST in an object class in
the hierarchy and a MAY in another object class in the hierarchy, the attribute type is
treated as a MUST. Referenced superior object classes must be defined in the schema.

ABSTRACT | STRUCTURAL | AUXILIARY
Indicates the type of object class. STRUCTURAL is the default.

MUST oids
List of one or more mandatory attribute types. Attribute types which are mandatory must
be specified when adding or modifying a directory entry.

MAY oids
List of one or more optional attribute types. Attribute types which are optional might
be specified when adding or modifying a directory entry.
```

Figure 4-4 objectclasses attribute format

4.5 Defining additional schema in IBM Tivoli Directory Server for z/OS

It might be necessary to define your own schema if there is nothing publicly available to represent the data that you want to store in the directory. Before defining additional schema, consider the following:

- ▶ What information should be stored in these entries, and what type of information is it?
- ▶ After deciding in the list of data to store in the entry, decide how the data should be represented in an LDAP entry. This is where the supported LDAP attribute syntaxes come

into play. Should the data be Integer syntax, Directory string syntax, Telephone Number syntax, Generalized Timestamp syntax, and so on? Other things to consider include whether the attribute type should be single or multi-valued in the directory, and the equality, ordering, and substring matching rules. The list of data obtained in these first two steps will likely become an attribute type in the LDAP server's schema.

- ▶ After you have the list of attribute types, you need to decide which attribute types are optional and which are required. This information can then be used to define your object class definitions.

4.5.1 Defining additional schema example

The following example shows how to define your own schema in IBM Tivoli Directory Server for z/OS. The attribute types and an object class in this example represent baseball position players as entries in IBM Tivoli Directory Server for z/OS. In these baseball position player entries, the first name, last name, hometown, position, hits, home runs, runs batted in (rbi), salary, and hobbies need to be stored.

The attribute types for first name, last name, hometown, position, and hobbies should have string representation in the directory. The hits, home runs, rbi, and salary should have integer representation in the directory because we want to be able to do ordering searches based on numeric representation instead of character representation.

For the string attribute types, determine the supported character data in these entries and the supported matching rules when searching or comparing the data in these entries. There are several attribute type string syntaxes with varying matching rules supported in IBM Tivoli Directory Server for z/OS such as: Bit string, Country string, Delivery method, Directory string, Enhanced Guide, Guide, IA5String, Numeric String, and Printable string. Before defining new attribute types in the schema, look at `schema.user.ldif`, `schema.IBM.ldif`, or other publicly defined schemas to see if there are any existing attribute types defined that would meet your directory needs. Taking a quick look at the schema shipped in the `schema.user.ldif` file, the `cn` (commonName) and `sn` (surName) attribute types could be used for the baseball player's first name and last name respectively. For the remaining attribute types, after analyzing the attribute type syntaxes and supported matching rules for these syntaxes, the Directory string syntax seems appropriate as it allows UTF-8 characters, which allows specifying multi-byte characters such as umlats in a person's hobbies. The equality matching rules supported with the Directory string syntax are **caseIgnoreMatch** and **caseExactMatching**, which provide flexibility while searching and comparing attribute values. After analyzing these strings, we decided that the equality matching rules should be **caseIgnoreMatch** because string case is not important while adding, modifying, searching, or comparing these attribute values. It was determined that the position and hobby attribute types should be multi-valued because baseball players are able to play multiple positions, and many have multiple hobbies.

For the hits, home runs, rbi, and salary attribute types, there are two integer type syntaxes supported in IBM Tivoli Directory Server for z/OS that could be used: Integer and Numeric String. For these attribute types, we chose Integer syntax because it depicts a single number, whereas the Numeric String is a blank separated list of numbers. Table 4-2 depicts the attribute types that will be used to represent a baseball position player entry.

Table 4-2 Attribute types example

Attribute types	Attribute Syntax	Equality matching rule	Ordering matching rule	Substring matching rule	Multi valued	Required or optional	Access class
cn	Directory string	caseIgnoreMatch	caseIgnoreOrderingMatch	caseIgnoreSubstringsMatch	N	Required	normal

Attribute types	Attribute Syntax	Equality matching rule	Ordering matching rule	Substring matching rule	Multi valued	Required or optional	Access class
sn	Directory string	caseIgnoreMatch	caseIgnoreOrderingMatch	caseIgnoreSubstringsMatch	N	Required	normal
hometown	Directory string	caseIgnoreMatch	caseIgnoreOrderingMatch	caseIgnoreSubstringsMatch	N	Required	normal
position	Directory string	caseIgnoreMatch	caseIgnoreOrderingMatch	caseIgnoreSubstringsMatch	Y	Required	normal
hits	Integer	integerMatch	N/A (will let default)	N/A (Substring matching rule does not apply to Integer syntax)	N	Required	normal
homeruns	Integer	integerMatch	N/A (will let default)	N/A (Substring matching rule does not apply to Integer syntax)	N	Required	normal
rbi	Integer	integerMatch	N/A (will let default)	N/A (Substring matching rule does not apply to Integer syntax)	N	Required	normal
salary	Integer	integerMatch	N/A (will let default)	N/A (Substring matching rule does not apply to Integer syntax)	N	Required	sensitive
hobby	Directory string	caseIgnoreMatch	caseIgnoreOrderingMatch	caseIgnoreSubstringsMatch	Y	Optional	normal

Note: When defining new attribute types or object classes in the schema, a unique and numeric object identifier (OID) should be specified. OIDs are strings of numbers separated by periods. OID “ranges” or “arcs” are allocated by naming authorities. You can get an OID arc for your company or organization by contacting the Internet Assigned Numbers Authority (IANA) at: <http://www.iana.org>. Search the site for “Private Enterprise Number” to apply for a Private Enterprise number. After you have obtained an OID arc, you can begin assigning OIDs to object classes and attribute types that you define.

The attribute types in Table 4-2 on page 83 must be put into an LDIF file so that the schema can be updated. Figure 4-5 on page 85 and Figure 4-6 on page 86 display the `schema.update.ldif` file that we used to modify the schema to add the new attribute types and the object class (`baseballPlayer`). The OID arc being used is the IBM OID arc, so do not use these OIDs for your own attribute type schema definitions.

```

dn: cn=schema
changetype: modify
replace: attributetypes
attributetypes: (
  1.3.18.0.2.1000.100.4.1
  NAME 'hometown'
  DESC 'The town where a person is from.'
  EQUALITY caseIgnoreMatch
  ORDERING caseIgnoreOrderingMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE
  USAGE userApplications
)
ibmattributetypes: (
  1.3.18.0.2.1000.100.4.1
  ACCESS-CLASS normal
)
attributetypes: (
  1.3.18.0.2.1000.100.4.2
  NAME 'position'
  DESC 'Baseball position'
  EQUALITY caseIgnoreMatch
  ORDERING caseIgnoreOrderingMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  USAGE userApplications
)
ibmattributetypes: (
  1.3.18.0.2.1000.100.4.2
  ACCESS-CLASS normal
)
attributetypes: (
  1.3.18.0.2.1000.100.4.3
  NAME 'hits'
  DESC 'Number of total hits'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE
  USAGE userApplications
)
ibmattributetypes: (
  1.3.18.0.2.1000.100.4.3
  ACCESS-CLASS normal
)
attributetypes: (
  1.3.18.0.2.1000.100.4.4
  NAME 'homeruns'
  DESC 'Number of homeruns hit'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE
  USAGE userApplications
)
ibmattributetypes: (
  1.3.18.0.2.1000.100.4.4
  ACCESS-CLASS normal
)

```

Figure 4-5 *schema.update.ldif*

```

attributetypes: (
  1.3.18.0.2.1000.100.4.5
  NAME 'rbi'
  DESC 'Number of runs batted in - rbi'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE
  USAGE userApplications
)
ibmattributetypes: (
  1.3.18.0.2.1000.100.4.5
  ACCESS-CLASS normal
)
attributetypes: (
  1.3.18.0.2.1000.100.4.6
  NAME 'salary'
  DESC 'The total salary/income of a person'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE
  USAGE userApplications
)
ibmattributetypes: (
  1.3.18.0.2.1000.100.4.6
  ACCESS-CLASS sensitive
)
attributetypes: (
  1.3.18.0.2.1000.100.4.7
  NAME 'hobby'
  DESC 'The hobby of a person'
  EQUALITY caseIgnoreMatch
  ORDERING caseIgnoreOrderingMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  USAGE userApplications
)
ibmattributetypes: (
  1.3.18.0.2.1000.100.4.7
  ACCESS-CLASS normal
)
-
add: objectclasses
objectclasses: (
  1.3.18.0.2.1000.100.6.1
  NAME 'baseballPlayer'
  DESC 'Used to represent a baseball player'
  SUP top
  STRUCTURAL
  MUST ( cn $ sn $ hometown $ position $ hits $ homeruns $ rbi $ salary )
  MAY ( hobby )
)
-

```

Figure 4-6 *schema.update.ldif continued*

The z/OS **ldapmodify** utility can be used to modify the schema:

```
ldapmodify -D adminDN -w passwd -f schema.update.ldif
```

adminDN is the **adminDN** specified in the LDAP server configuration file and *passwd* is the password for the LDAP administrator's entry.

4.6 Defining additional schema for use with RACF custom fields

The minimum schema contains the attribute types that are necessary to use the SDBM back end without applying any additional schema. Each RACF command line keyword is mapped to an LDAP attribute type. For example, a RACF user's OMVS segment UID keyword is mapped to the LDAP attribute type **racf0mvsUid**. See the Accessing RACF information chapter in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for all of the RACF keyword and LDAP attribute type mappings. However, if there are RACF users or groups that have a RACF custom field defined in the CSDATA segment, the minimum schema must be updated to map the custom field to an LDAP attribute type. This allows this data to be added, modified, and displayed on searches.

Similar to updating the schema for other new LDAP attribute types, an **attributetypes** and **ibmattributetypes** value must be specified. However, the **ibmattributetypes** attribute type format contains an optional RACFFIELD parameter that specifies the appropriate RACF user or group CSDATA segment to map this LDAP attribute type to.

See sections **Attributetypes** and **ibmattributetypes** attribute format in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for the format of the **attributetypes** and **ibmattributetypes** values.

Figure 4-7 on page 88 shows the `custom.field.ldif` file we used to modify the schema to add two new attribute types to map the `USER.CSDATA.WRKPHNE` and `GROUP.CSDATA.WRKGRP` custom fields to LDAP attribute types. Both custom fields are defined as character fields in RACF. The OID arc being used is the IBM OID arc, so do not use these OIDs for your own attribute type schema definitions.

```

dn: cn=schema
changetype: modify
replace: attributetypes
attributetypes: (
  1.3.18.0.2.1000.100.4.10
  NAME 'racfworkphone'
  DESC 'Represents the WRKPHNE field in the RACF user CSDATA segment'
  EQUALITY caseIgnoreMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
  SINGLE-VALUE
  USAGE userApplications
)
attributetypes: (
  1.3.18.0.2.1000.100.4.11
  NAME 'racfworkgroup'
  DESC 'Represents the WRKGRP field in the RACF group CSDATA segment'
  EQUALITY caseIgnoreMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
  SINGLE-VALUE
  USAGE userApplications
)
-
add: ibmattributetypes
ibmattributetypes: (
  1.3.18.0.2.1000.100.4.10
  ACCESS-CLASS sensitive
  RACFFIELD ('USER-CSDATA-WRKPHNE' 'char')
)
ibmattributetypes: (
  1.3.18.0.2.1000.100.4.11
  ACCESS-CLASS sensitive
  RACFFIELD ('GROUP-CSDATA-WRKGRP' 'char')
)

```

Figure 4-7 *custom.field.ldif* example

The z/OS **ldapmodify** utility can be used to modify the schema to add these two attribute types:

```
ldapmodify -D adminDN -w passwd -f custom.field.ldif
```

adminDN is the **adminDN** specified in the LDAP server configuration file and *passwd* is the password for the LDAP administrator's entry.



Authentication, authorization, and security

This chapter describes issues of authentication, authorization, and security.

5.1 Overview

Needless to say, IT security is of the utmost importance for most, if not all organizations that use IT in their day-to-day operations. Effective IT security requires mechanisms to implement, among others, the following crucial concepts to secure critical resources and data:

- ▶ Authentication
- ▶ Authorization
- ▶ Confidentiality
- ▶ Audit

These concepts can be implemented by separate software components that can be architected into a security engine for the enterprise. Or alternatively, IBM Tivoli Directory Server for z/OS can be used as the security engine for an enterprise. Moreover, for organizations that have invested in RACF as the security engine for System z® specific critical resources and data, IBM Tivoli Directory Server for z/OS allows for RACF control within software components that are not compatible or interoperable with RACF. The focus of this section will be on IBM Tivoli Directory Server for z/OS's security features and capabilities including its RACF synergy.

Before describing IBM Tivoli Directory Server for z/OS's security features, the key security concepts should be defined in general terms:

Authentication	The process of verifying a Subject's identity
Authorization	The process of giving a Subject permission to perform a requested Action against a protected Object
Confidentiality	The guarantee that unauthorized Subjects do not have access to a protected Object
Audit	The process of documenting Actions done against protected Objects by Subjects

The LDAP Internet drafts, RFC documentation, and IBM Tivoli Directory Server for z/OS documentation use different terminology than the terms listed above when describing these concepts. Nevertheless, the semantics defined in the general definition are equivalent to the features described in the protocol and implemented in IBM Tivoli Directory Server. In later sub-sections we will highlight IBM Tivoli Directory Server for z/OS's implementation of each general concept.

The reason for beginning the security discussion in abstract terms is that there are slightly different perspectives when it comes to IBM Tivoli Directory Server's implementation of the key security concepts. First, IBM Tivoli Directory Server for z/OS is a repository of data, and the data itself might need to be secured. Next, the LDAP protocol's remote nature and IBM Tivoli Directory Server for z/OS features, especially when considering its RACF synergy, allow it to participate in securing resources that aren't necessarily represented in its data repository.

Table 5-1 on page 91 shows the mapping of general IT security concepts to IBM Tivoli Directory Server for z/OS features. Data stored in an LDAP directory can vary, but some examples highlight the security implications. For instance, LDAP directories can consist of LDAP entries that represent staff members of an organization. These entries might have attribute values that must be secured, such as a person's Social Security Number in the US. In this use case, IBM Tivoli Directory Server must secure its contents. It uses binds, ACLs, SSL, and activity log to secure the data from unauthorized searches, reads, writes, and compares.

Table 5-1 Tivoli Directory Server - Data repository

General Concept	Tivoli Directory Server Equivalent
Authentication	Bind
Subject	Client, bind DN RACF ID (SDBM/Native Authentication)
Object	Entry RACF entities: USERS, GROUPS, RESOURCE PROFILES (SDBM)
Authorization	ACL RACF access check (SDBM)
Confidentiality	Encrypted communication (SSL/TLS) Encrypted passwords
Audit	SMF audit, Activity Logs
Action	read, write, search, compare or object add/delete RACF access: ALTER, READ, CONTROL, UPDATE (SDBM)

On the other hand, Table 5-2 shows the equivalents in a RACF-backed centralized security engine for heterogeneous environments.

Table 5-2 Tivoli Directory Server - Security Decision Engine (RACF/ICTX plug-in)

General Concept	Tivoli Directory Server Equivalent
Authentication	Bind
Subject	RACF ID
Object	RACF resource profile
Authorization	RACF access check
Confidentiality	RACF PROTECTED
Audit	SMF audit
Action	ALTER, READ, CONTROL, UPDATE

Let's look at how this software configuration can secure the enterprise by focusing on authentication, authorization, and audit.

Authentication

A configured IBM Tivoli Directory Server for z/OS server with or without RACF could handle all authentication within an enterprise, regardless if a user requires access to protected resources.

For example, Linux workstation logins can be implemented by having a Linux PAM/NSS module configured to use IBM Tivoli Directory Server for z/OS. When the user attempts to log in by entering their credentials (username/password), the credentials (after mapping) are used to bind to IBM Tivoli Directory Server. If the bind succeeds, workstation access is granted. If the mapping of the credentials lead to an SDBM bind DN, or if native authentication is set up in IBM Tivoli Directory Server, RACF will ultimately authenticate the user. A key feature of IBM Tivoli Directory Server for z/OS R12 and later, and RACF, is the ability to enforce password policies.

Commonly, organization set password policies to help reduce the risk of systems being compromised. Later we discuss IBM Tivoli Directory Server and password policy in more detail.

Authorization and Audit

A configured IBM Tivoli Directory Server for z/OS server with the ICTX plug-in will accept LDAP extended operation requests to go to RACF for profile access checks and to issue SMF audit records. The ICTX plug-in is a configuration option for IBM Tivoli Directory Server for z/OS that handles special extended operation LDAP requests that result in RACF interaction.

Basically, application writers can write applications in C/C++ or Java and use widely available LDAP client APIs to use RACF to secure resources throughout the enterprise by providing an authorization engine and remote manner for issuing SMF audit records. Some RACF knowledge is required. However, the LDAP protocol handles communication between RACF and non-z/OS applications.

Now that we have given an overview of IT security and the role IBM Tivoli Directory Server for z/OS can play in the enterprise, we will discuss in more detail all of IBM Tivoli Directory Server for z/OS's security related features.

5.2 Authentication mechanisms supported by IBM Tivoli Directory Server for z/OS

Authentication request operations are used to establish and end a session between an LDAP client and server. The following terms are used to indicate the type of authentication related operations:

Bind	Initiates the connection between the client and a server. This allows the client to prove its identity by authenticating itself to the server.
Unbind	Terminate a client/server session.
Abandon	Allows a client to request that the server abandon an outstanding operation.

The LDAP RFCs (Request for Comments) specify a number of authentication or bind mechanisms that are supported by LDAP servers. As of z/OS V1R12, IBM Tivoli Directory Server for z/OS supports the following bind or authentication mechanisms:

- ▶ Anonymous
- ▶ Simple
- ▶ CRAM-MD5
- ▶ DIGEST-MD5
- ▶ GSS-API (Kerberos)
- ▶ EXTERNAL (SSL)

CRAM-MD5, DIGEST-MD5, GSSAPI, and EXTERNAL binds are types of Simple Authentication and Security Layer (SASL) authentication mechanisms as documented in RFC 2222 (<http://www.ietf.org/rfc/rfc2222.txt>). Each LDAP bind mechanism has a different method of allowing the client to prove its identity using credentials (username and password) or other means. It is also possible to secure the connection between the client and server by using SSL/TLS. See 5.8, "SSL/TLS" on page 133 for more information about configuring SSL/TLS.

5.2.1 Anonymous

By default, IBM Tivoli Directory Server for z/OS allows anonymous or unauthenticated access to the directory. As the name applies, an LDAP client application is automatically anonymously bound to the LDAP server when not specifying any credentials (username and password) on a bind request. However, usually authorization to perform LDAP requests is limited when anonymously authenticated unless authorization has been granted to `cn=anybody` in ACLs. See 5.4, “Authorization using Tivoli Directory Server Access Control Lists (ACL)” on page 108 for more information about authorization.

By default with the IBM Tivoli Directory Server for z/OS, anonymous users are allowed to search the root DSE and the `cn=monitor` entries. The `rootDSE` retrieves general information about the LDAP server such as the configured suffixes, and supported and enabled capabilities of the LDAP server. A `cn=monitor` search retrieves statistics such as the number of add, search, compare, and modify requests that have occurred to each configured back end. If the `-D` (bindDN) and `-w` (password) options are not specified on the z/OS LDAP client utilities, an anonymous bind is performed. See *IBM Tivoli Directory Server Client Programming for z/OS V1R12.0*, SA23-2214-04 for more information about the z/OS LDAP client utilities such as `ldapsearch`.

For example to search the `rootDSE` with the `ldapsearch` utility while bound anonymously:

```
ldapsearch -s base -b "" "objectclass=*"
```

To search the `cn=monitor` entries with the `ldapsearch` utility while bound anonymously:

```
ldapsearch -s sub -b "cn=monitor" objectclass=*
```

However if security regulations require that anonymous or unauthenticated access is not allowed to IBM Tivoli Directory Server for z/OS, the `allowAnonymousBinds off` configuration option can be specified in the global section of the LDAP server configuration file. By default, anonymous binds are allowed to IBM Tivoli Directory Server for z/OS.

5.2.2 Simple

One of the most common authentication mechanisms everyone is familiar with is a simple bind. An LDAP simple bind is performed by specifying a distinguished name (DN) and a password value on an LDAP bind request.

If the DN specified on the bind request falls under a configured LDBM, TDBM, or CDBM suffix and the entry has a `userpassword` attribute value, the password verification that is performed depends on how the `userpassword` attribute value is encrypted or hashed:

- ▶ If the entry's `userpassword` attribute value is encrypted in AES or DES, the password value in the entry is decrypted and checked against the password value specified on the bind request. If the values are the same, then authentication is successful.
- ▶ If the entry's `userpassword` attribute value is hashed in a one-way hash, such as crypt, md5, SHA, or SSHA (Salted SHA-1 – only in z/OS V1R12 and later), the password specified on the bind request is hashed in the same algorithm. If the two hashes are the same, then authentication is successful.
- ▶ If the DN specified on the bind request falls under the configured SDBM suffix, the password checking is performed by RACF by passing the value on the bind request to RACF.
- ▶ If the DN specified on the bind request falls under a configured LDBM, TDBM, or CDBM suffix and the entry resides under a native authentication subtree and is participating in native authentication, the configured z/OS security manager performs the password

verification checking. The password specified on the bind request is sent to the security manager by issuing `__passwd()` or `RACROUTE REQUEST=VERIFY`.

With the z/OS LDAP client utilities, a simple bind is performed by specifying the `-D` (bindDN) and `-w` (password) options.

This example illustrates that distinguished name (DN), `cn=jon,o=ibm,c=us`, is attempting a simple bind with password secret:

```
ldapsearch -D "cn=jon,o=ibm,c=us" -w secret -s base -b "cn=jon,o=ibm,c=us"
"objectclass=*"
```

One of the limitations with simple binds is that the password is exposed in the clear. However, this problem can be rectified by using a secure SSL/TLS connection between the LDAP server and the client application to hide the distinguished name and password while the request is in transit. See 5.8, "SSL/TLS" on page 133 for more information about using SSL/TLS between IBM Tivoli Directory Server for z/OS and your LDAP client applications.

5.2.3 CRAM-MD5

To help overcome one of the inherent weaknesses of simple binds (the password being exposed in the clear on the bind request while in transit), the CRAM-MD5 bind mechanism was developed in RFC 2195 (<http://www.ietf.org/rfc/rfc2222.txt>). There are a few advantages of using the CRAM-MD5 bind mechanism:

1. The password value is no longer transmitted in the clear on the bind request. The password entered in the LDAP client application is hashed with a random string returned from the LDAP server when a CRAM-MD5 bind request is wanted. By hashing the password value in this manner, it is no longer transmitted in the clear.
2. Like the simple bind, CRAM-MD5 allows a distinguished name (DN) and password value to be specified on the bind request. However, you can also authenticate using a short unique username that exists in an TDBM, LDBM, or CDBM entry as a uid attribute value. This makes it easier for users because they do not need to know the underlying full distinguished name (DN) when authenticating to IBM Tivoli Directory Server for z/OS.

The CRAM-MD5 bind involves multiple handshakes between the LDAP client application and LDAP server to help hash the clear text password. This helps to disguise the password while it is in transit between the client and server.

Depending on your LDBM, TDBM, and CDBM entry setup, there are multiple ways of performing a CRAM-MD5 bind:

1. Specify a bind distinguished name (DN) on the bind request that resides under a configured LDBM, TDBM, or CDBM suffix and a password value. This is the same as performing a simple bind.
2. Specify a username and password value on the bind request. The username that is specified must exist as a uid attribute value in an entry that resides under a configured LDBM, TDBM, or CDBM suffix.
3. Specify a bind DN, username, and password value on the bind request. The username that is specified must exist as an uid attribute value in the entry pointed to by the distinguished name that is specified.

Note the following:

- a. CRAM-MD5 binds are only supported to entries that reside in the LDBM, TDBM, and CDBM back ends.

- b. If you are planning on using CRAM-MD5 binds, the **userpassword** attribute values must either be encrypted in AES or DES or not encrypted at all. The LDAP server must be able to get clear access to the password value to allow CRAM-MD5 authentications to successfully occur.

Figure 5-1 shows entries in the LDBM and TDBM back ends that are used in the following CRAM-MD5 bind examples.

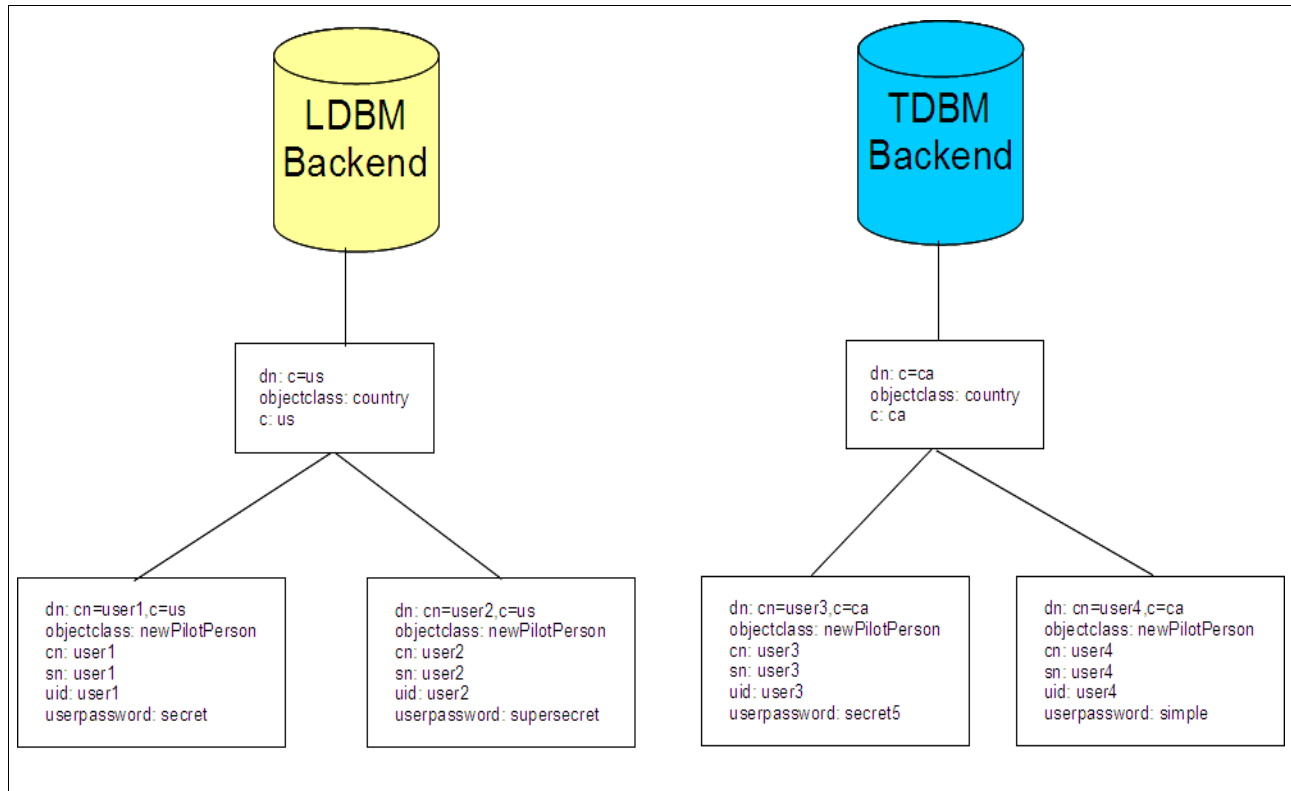


Figure 5-1 Example entries used for CRAM-MD5 binds

The newPilotPerson objectclass is present in the schema.user.ldif file and has the uid attribute as an optional attribute type. Any objectclass that allows the uid attribute to be specified in an entry can be used to perform CRAM-MD5 binds.

1. This example uses the z/OS **ldapsearch** client utility and performs a CRAM-MD5 bind to user1 with password of secret. The username is specified on the **-U** option of the z/OS LDAP client utilities.

```
ldapsearch -m CRAM-MD5 -U user1 -w secret -b "cn=user1,c=us" "objectclass=*"

```

The LDAP server does a search of the LDBM and TDBM back ends looking for an entry that has a uid attribute of user1 and finds the cn=user1,c=us entry in the LDBM back end. The password value is then checked to verify that the correct password was entered on the z/OS LDAP client utility.

2. This example uses the z/OS **ldapsearch** client utility and performs a CRAM-MD5 bind with bind distinguished name (DN), cn=user2,c=us, and a password of supersecret.

```
ldapsearch -m CRAM-MD5 -D "cn=user2,c=us" -w supersecret -b "cn=user2,c=us"
"objectclass=*"

```

Because only a distinguished name (DN) was specified on the z/OS LDAP client utility while attempting to perform an CRAM-MD5 bind, the LDAP server finds the distinguished name in the LDBM back end and verifies that the correct password was specified.

3. This example uses the z/OS **ldapsearch** client utility and performs a CRAM-MD5 bind with bind distinguished name (DN), cn=user3,c=ca, username, user3, and a password of secret5.

```
ldapsearch -m CRAM-MD5 -D "cn=user3,c=ca" -U user3 -w secret5 -s base -b
"cn=user3,c=ca" "objectclass=*"
```

Because a bind distinguished name (DN) and username is specified, the DN is found in the TDBM back end and then a check is done to ensure the entry has a uid attribute value of user3, which it does. After that is verified, the password is checked to verify that it is correct.

4. This example uses the z/OS **ldapsearch** utility and attempts to perform a CRAM-MD5 bind with bind distinguished name (DN), cn=user4,c=ca, username nothere, and a password of simple. However, this authentication is not successful because cn=user4,c=ca does not have an uid attribute value of nothere.

```
ldapsearch -m CRAM-MD5 -D "cn=user4,c=ca" -U nothere -w simple -s base -b
"cn=user4,c=ca" "objectclass=*"
```

Returns:

```
ldap_sasl_bind: Inappropriate authentication
ldap_sasl_bind: additional info: R004112 A bind argument is not valid
(tdbm_get_user_password)
```

5.2.4 DIGEST-MD5

Shortly after the CRAM-MD5 bind mechanism was introduced, the DIGEST-MD5 bind mechanism was introduced in RFC 2831, <http://www.ietf.org/rfc/rfc2195.txt>, to allow for integrity and confidentiality protection on LDAP requests after a successful authentication. This support was lacking in the CRAM-MD5 authentication mechanism. The integrity and confidentiality protection is used to encrypt the data exchanged between the client and server until the authenticated user unbinds or disconnects from the LDAP server.

Like the CRAM-MD5 bind mechanism, the DIGEST-MD5 bind mechanism involves multiple handshakes between the LDAP client application and LDAP server to help hash the clear text password. However, with the DIGEST-MD5 bind mechanism, additional data is exchanged between the client and server to perform a better hash of the clear text password.

Depending on your LDBM, TDBM, and CDBM entry setup, there are multiple ways of performing a DIGEST-MD5 bind:

1. Specify a username and password value on the bind request. The username that is specified must exist as a uid attribute value in an entry that resides under a configured LDBM, TDBM, or CDBM suffix.
2. Specify a bind DN, username, and password value on the bind request. The username that is specified must exist as an uid attribute value in the entry pointed to by the distinguished name that is specified.

Note the following:

1. Unlike a CRAM-MD5 bind request, the username must be specified on the DIGEST-MD5 bind request. Therefore only specifying the bind distinguished name (DN) on the DIGEST-MD5 bind request is not supported.
2. DIGEST-MD5 binds are only supported to entries that reside in the LDBM, TDBM, and CDBM back ends.

- If you are planning on using DIGEST-MD5 binds, the `userpassword` attribute values must either be encrypted in AES or DES, or not encrypted at all. The LDAP server must be able to get clear access to the password value to allow DIGEST-MD5 authentications to successfully occur.

Figure 5-2 shows entries in the LDBM and TDBM back ends that are used in the following DIGEST-MD5 bind examples.

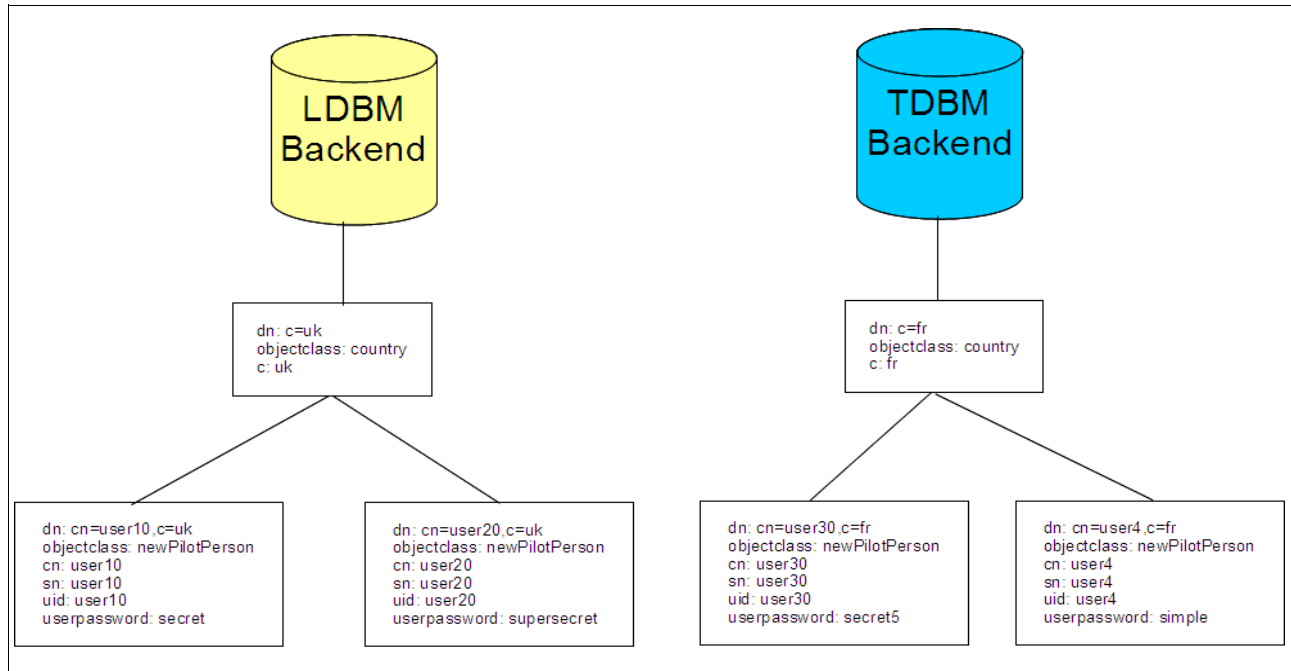


Figure 5-2 Example entries used for DIGEST-MD5 binds

The `newPilotPerson` objectclass is present in the `schema.user.ldif` file and has the `uid` attribute as an optional attribute type. Any **objectclass** that allows the `uid` attribute to be specified in an entry can be used to perform DIGEST-MD5 binds.

- This example uses the z/OS `ldapsearch` client utility and performs a DIGEST-MD5 bind to `user10` with password of `secret`. The username is specified on the `-U` option of the z/OS LDAP client utilities.

```
ldapsearch -m DIGEST-MD5 -U user10 -w secret -b "cn=user10,c=uk"
"objectclass=*"

```

The LDAP server does a search of the LDBM and TDBM back ends looking for an entry that has an `uid` attribute of `user10` and finds the `cn=user10,c=uk` entry in the LDBM back end. The password value is then checked to verify that the correct password was entered on the z/OS LDAP client utility.

- This example uses the z/OS `ldapsearch` client utility and performs a DIGEST-MD5 bind with bind distinguished name (DN), `cn=user30,c=fr`, username, `user30`, and a password of `secret5`.

```
ldapsearch -m DIGEST-MD5 -D "cn=user30,c=fr" -U user30 -w secret5 -s base -b
"cn=user30,c=fr" "objectclass=*"

```

Because a bind distinguished name (DN) and username is specified, the DN is found in the TDBM back end and then a check is done to ensure the entry has a `uid` attribute value of `user10`, which it does. After that is verified, the password is checked to verify that it is correct.

3. This example uses the z/OS **ldapsearch** utility and attempts to perform a DIGEST-MD5 bind with a bind distinguished name (DN), cn=user4,c=fr, username nothere, and a password of simple. However, this DIGEST-MD5 authentication is not successful because cn=user4,c=fr does not have a uid attribute value of nothere.

```
ldapsearch -m DIGEST-MD5 -D "cn=user4,c=fr" -U nothere -w simple -s base -b  
"cn=user4,c=fr" "objectclass=*"
```

Returns:

```
ldap_sasl_bind: Inappropriate authentication  
ldap_sasl_bind: additional info: R004112 A bind argument is not valid  
(tdbm_get_user_password)
```

5.2.5 GSS-API (Kerberos)

IBM Tivoli Directory Server for z/OS allows clients to authenticate to the server by using the IBM Network Authentication and Privacy Service, better known as Kerberos Version 5. Kerberos is a trusted third party, private-key, network authentication system. In Kerberos, a ticket (a packet of information used by a client to prove its identity) is passed to a server in place of a user name and password. This ticket is encrypted and cannot be duplicated. After the server verifies the client ticket, it sends its own ticket to the client so the client to authenticate it. After the mutual authentication process is complete, the client and server have authenticated each other. See Figure 5-3 for a diagram of the LDAP server and Kerberos setup.

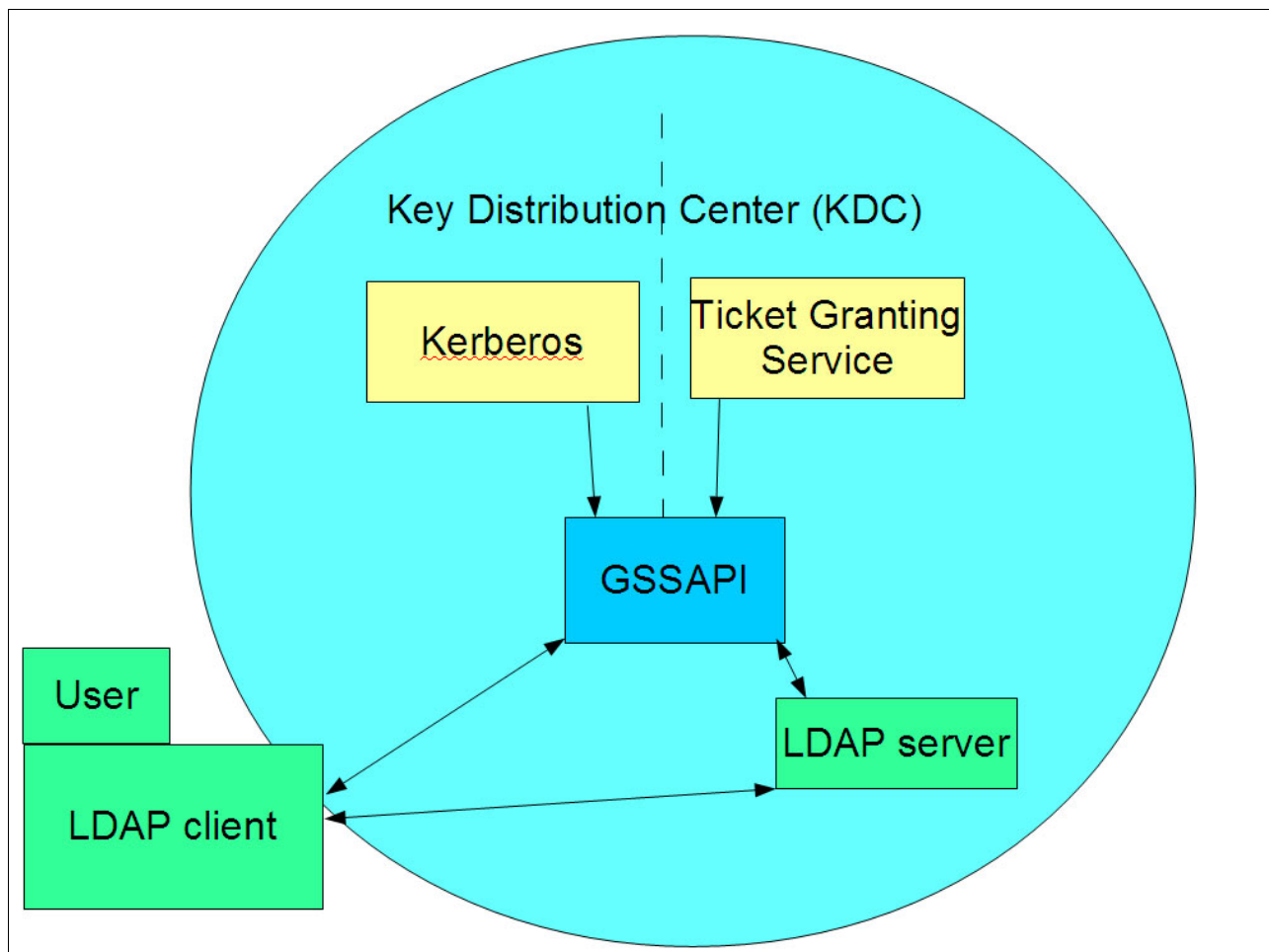


Figure 5-3 LDAP-KERBEROS setup

IBM Tivoli Directory Server for z/OS supports Kerberos integrity and confidentiality services. Upon successful completion of a SASL bind operation using the GSS API mechanism, the negotiated quality of protection (QOP) is used for subsequent messages sent over the connection. This QOP continues to be used until the completion of a new SASL bind request. If the new SASL bind request fails, the connection reverts to anonymous authentication with no integrity or confidentiality services. See *Understanding LDAP design and implementation*, SG24-4986 for additional information about Kerberos on z/OS.

As documented in the Kerberos authentication chapter of *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05, there are a number of Kerberos related configuration options in the global and back end specific sections of the LDAP server configuration file to fully take advantage of Kerberos authentication.

The following are the Kerberos related configuration options that belong in the global section of the LDAP server configuration file:

- ▶ **supportKrb5**: This option activates Kerberos authentication in the LDAP server
- ▶ **serverKrbPrinc**: This option specifies the Kerberos principal name assigned to the LDAP server.
- ▶ **krbKeytab**: This option specifies the Kerberos key table that is used by the LDAP server.
- ▶ **krbLDAPAdmin**: This option specifies the Kerberos identity that represents an LDAP administrator. It is equivalent to the **adminDN** option in the LDAP server configuration file.

If **krbIdentityMap on** is specified in the TDBM, LDBM, SDBM, or CDBM back ends, additional Kerberos identity mapping is performed.

Assume that the LDAP server configuration file has the following Kerberos-related configuration options specified. Note that the configuration file in Figure 5-4 on page 100 does not have all required configuration options present.

```
# Global Section
supportKrb5 on
serverKrbPrinc LDAP/ibm.com@IBM.COM
krbLDAPAdmin ibm-kn=ldapadm@IBM.COM
krbKeytab none

# TDBM Section
database tdbm GLDBTD31
suffix o=Lotus,c=us
krbIdentityMap on

# LDBM Section
database ldbm GLDBLD31/GLDBLD64
suffix o=ibm,c=us
krbIdentityMap on

# SDBM Section
database sdbm GLDBSD31/GLDBSD64
suffix sysplex=plex1
krbIdentityMap on

# CDBM Section
database cdbm GLDBCD31/GLDBCD64
krbIdentityMap on
```

Figure 5-4 Partial LDAP server configuration file with Kerberos options

Our example uses the configuration file in Figure 5-4. Before performing a Kerberos (GSSAPI) bind with the z/OS LDAP client utilities, the RACF user ID, JEFF, is updated to specify a KERB segment with a KERBNAME value:

```
ALTUSER JEFF KERB(KERBNAME(JEFF))
```

JEFF issues a Kerberos **kinit** command to acquire a Kerberos ticket:

```
kinit JEFF
EUVF06017R Enter password:
```

After issuing the **kinit** command, the z/OS LDAP client utilities can be invoked to perform a Kerberos bind with identity `jeff@IBM.COM` with the ticket retrieved from the Kerberos Distribution Center (KDC). `IBM.COM` is the name of the Kerberos realm.

```
ldapsearch -m GSSAPI -b "cn=jeff,c=us" "objectclass=**"
```

During the bind process in the LDAP server, the Kerberos identity `jeff@IBM.COM` by default is mapped to a bind distinguished name (DN) of `ibm-kn=jeff@IBM.COM`. Because Kerberos identity mapping is configured in the SDBM back end and RACF user ID JEFF has a KERBNAME value of JEFF, the `racfid=JEFF,profiletype=user,sysplex=plex1` distinguished name is added to what is called the alternate DN list. Because Kerberos identity mapping is

also configured in the LDBM, TDBM, and CDBM back ends, identity mapping is performed in those back ends and any distinguished names found are also added to the alternate DN list.

The bind DN and alternate DNs can be used in ACLs (Access Control Lists) for authorization to resources in the LDBM, TDBM, or CDBM back ends. See 5.4, “Authorization using Tivoli Directory Server Access Control Lists (ACL)” on page 108 for more information about authorizing users to resources. See the Identity mapping section in the Kerberos authentication chapter of *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for information about how Kerberos identity mapping is performed in the LDBM, TDBM, SDBM, and CDBM back ends.

5.2.6 External (SSL)

The IBM Tivoli Directory Server for z/OS supports the SASL EXTERNAL (SSL) bind mechanism. This means that the authentication on the bind is performed using the data obtained during the SSL/TLS client authentication that was performed on the SSL/TLS handshake with the client.

To enable SASL EXTERNAL binds in IBM Tivoli Directory Server for z/OS, the following updates are necessary:

1. **sslAuth serverClientAuth**: Must be specified so that the server can authenticate the client.
2. **sslKeyRingFile file**: Must specify a valid SSL key database file, RACF keyring, or PKCS #11 token

For more information about configuring SSL/TLS in IBM Tivoli Directory Server for z/OS, see 5.8, “SSL/TLS” on page 133.

The client connects to the LDAP server and performs the SSL/TLS handshake. The handshake sends the client certificate to the LDAP server.

The client performs a SASL bind with a bind mechanism of EXTERNAL (-m option) on the z/OS LDAP client utilities and specifies the SAF key ring, SSL key database file, or PKCS#11 token, and specifies the certificate label if using a certificate or token other than the default.

At this point, the LDAP server considers the bind DN of the client for authorization purposes to be the client’s DN as transmitted in the client’s certificate on the handshake. The name specified in the BIND request must either match the subject name in the client certificate or be null.

In z/OS V1R10, IBM Tivoli Directory Server for z/OS added support to allow certificates on SASL EXTERNAL binds to optionally be mapped to RACF users. The RACF RACDCERT MAP command is used to associate a certificate to a RACF user ID. The **sslMapCertificate** configuration option in the global section of the LDAP server configuration file indicates the user ID mapping (if any) that is to be performed:

```
sslMapCertificate {off | check | add | replace} {fail | ignore}
```

When **check**, **add**, or **replace** is specified for the first value, RACF is searched for the user ID associated with the certificate used during a SASL EXTERNAL bind. The **sslKeyRingFile** configuration option must be specified to indicate which key database, RACF key ring, or PKCS#11 token to use to do this. If there is no RACF user ID associated with the certificate and **fail** is specified for the second value, the SASL EXTERNAL bind fails. If there is no associated RACF user ID and **ignore** is specified for the second value, the bind continues without mapping the certificate to a RACF user.

If an associated RACF user ID is found and add or replace is specified for the first value, a distinguished name (DN) is created based on the user ID and the SDBM suffix. For **add**, this mapped DN is added to the alternate DN list associated with the bind DN that was created from the subject's name in the certificate. For **replace**, this mapped DN replaces the bind DN that was created from the subject's name in the certificate. The mapped DN is used when gathering the groups in which the bound user exists, and when checking authorization for LDAP operations, including SDBM operations. The SDBM back end must be configured when **add** or **replace** is specified.

When **off** is specified for the first value, RACF is not searched for the user ID associated with the certificate and no certificate mapping is performed. In this case, it does not matter what the second value is.

The default values for the **sslMapCertificate** configuration option is **off fail**.

The following are examples of using the RACDCERT MAP command to map certificates to existing RACF users:

1. Map a certificate that has a Subject's name of CN=JON.O=IBM.C=US to RACF user ID JONC with a label of JONCMAP:

```
RACDCERT ID(JONC) MAP SDNFILTER('CN=JON.O=IBM.C=US') WITHLABEL('JONCMAP')
```

2. Map any certificate that has O=IBM and C=CA in the Subject's name to RACF user ID CAUSER with a label of CAUSERMAP:

```
RACDCERT ID(CAUSER) MAP SDNFILTER('O=IBM.C=CA') WITHLABEL('CAUSERMAP')
```

3. Map all certificates that are issued by VeriSign for Class 1 Individual Subscribers with a label of VERISIGNMAP to RACF user ID GUSER:

```
RACDCERT ID(GUSER) MAP  
IDNFILTER('OU=VeriSign Class 1 Individual Subscriber.O=VeriSign,  
Inc..L=Internet') WITHLABEL('VERISIGNMAP')
```

Note the following:

1. The RACF user that issues the RACDCERT MAP command must have RACF SPECIAL authority.
2. The RACF user ID that is being mapped to must have an OMVS segment defined in order for IBM Tivoli Directory Server for z/OS to successfully perform the mapping.
3. If the DIGTNMAP or DIGTCRIT classes are RACLISTed, refresh the classes to activate your changes after performing the RACDCERT MAP command:

```
SETROPTS RACLIST(DIGTNMAP, DIGTCRIT) REFRESH
```

See *z/OS V1R12.0 Security Server RACF Command Language Reference*, SA22-7687 for more information about using the RACDCERT MAP command.

The following examples illustrate how the bind DN and alternate DN are set when performing a SASL EXTERNAL bind with different **sslMapCertificate** option settings. These examples assume that an SDBM back end is configured with a suffix of `cn=racf` and the certificate mappings above are used.

This example performs a SASL EXTERNAL bind with SAF keyring, LDAPCLIENT and SSL certificate with a label of CLIENT. The CLIENT certificate has a Subject's Name of CN=JON.O=IBM.C=US and an Issuer's Name of CN=SELSIGN.O=IBM.C=US.

```
ldapsearch -Z -K LDAPCLIENT -N CLIENT -m EXTERNAL -s base -b "" "objectclass=*"
```

Table 5-3 SASL EXTERNAL mapping example

sslMapCertificate values	Bind distinguished name (DN)	Alternate distinguished name (DN)
off fail	CN=JON,O=IBM,C=US	N/A
add fail	CN=JON,O=IBM,C=US	RACFID=JONC,PROFILETYPE=USER,CN=RACF
replace fail	RACFID=JONC,PROFILETYPE=USER,CN=RACF	N/A

This example performs a SASL EXTERNAL bind with SSL certificate client in key database file, /home/user/key.kdb, that has a password value of **secret**. The client certificate has a Subject's Name of CN=JEFF.O=IBM.C=CA and an Issuer's Name of CN=SELFSIGN.O=IBM.C=CA.

```
ldapsearch -Z -K /home/user/key.kdb -N client -P secret -m EXTERNAL -s base -b "" "objectclass=*"
```

Table 5-4 SASI EXTERNAL mapping example

sslMapCertificate values	Bind distinguished name (DN)	Alternate distinguished name (DN)
off fail	CN=JEFF,O=IBM,C=CA	N/A
add fail	CN=JEFF,O=IBM,C=US	RACFID=CAUSER,PROFILETYPE=USER,CN=RACF
replace fail	RACFID=CAUSER,PROFILETYPE=USER,CN=RACF	N/A

5.3 Native authentication

As described in 5.2, “Authentication mechanisms supported by IBM Tivoli Directory Server for z/OS” on page 92, IBM Tivoli Directory Server for z/OS simple binds or authentications are supported to the TDBM, LDBM, CDBM, or SDBM back ends when a distinguished name and a password value are specified on the bind request. When attempting a simple bind to an entry residing in the TDBM, LDBM, or CDBM back ends, the password value must be specified as a **userPassword** attribute value in the entry itself. IBM Tivoli Directory Server for z/OS does the password verification itself to determine if the simple bind or authentication is successful. Figure 5-5 illustrates an LDAP client application performing a simple bind to an entry residing in the TDBM, LDBM, or CDBM back end. First the application does an LDAP search with a base distinguished name (DN) of `o=ibm,c=us` and a filter of `cn=jonc` to find the entry in the directory. If the entry is found, the LDAP client application then performs a simple bind or authentication as that entry by specifying a password of `joncldap`.

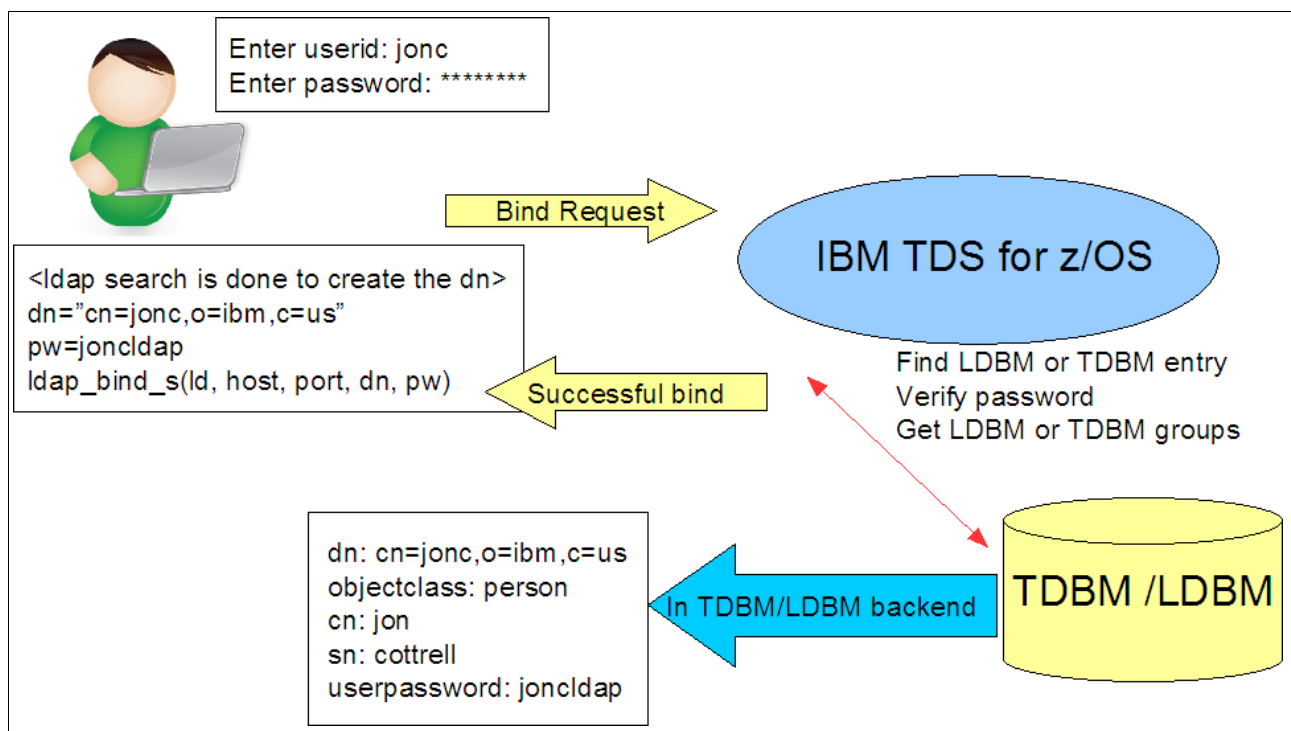


Figure 5-5 TDBM and LDBM simple bind example

Simple binds to the SDBM back end are also supported. In this case, the password or password phrase is stored in the RACF database for the specified RACF user. When a simple bind request is done with a user that resides in the SDBM back end, IBM Tivoli Directory Server for z/OS does an `__passwd()` or `RACROUTE REQUEST=VERIFY` with the user ID and password or password phrase that was specified on the bind request. RACF does the verification of the user's password or password phrase on the bind request. Figure 5-6 on page 105 illustrates an LDAP client application performing a simple bind to an entry residing in the SDBM back end. First the application does an LDAP search with a base distinguished name (DN) of `profilotype=user,cn=sdbm` and a filter of `racfid=u1234` to find the entry in the directory. If the entry is found, the LDAP client application then performs a simple bind or authentication as that entry by specifying a password of `racfpw`.

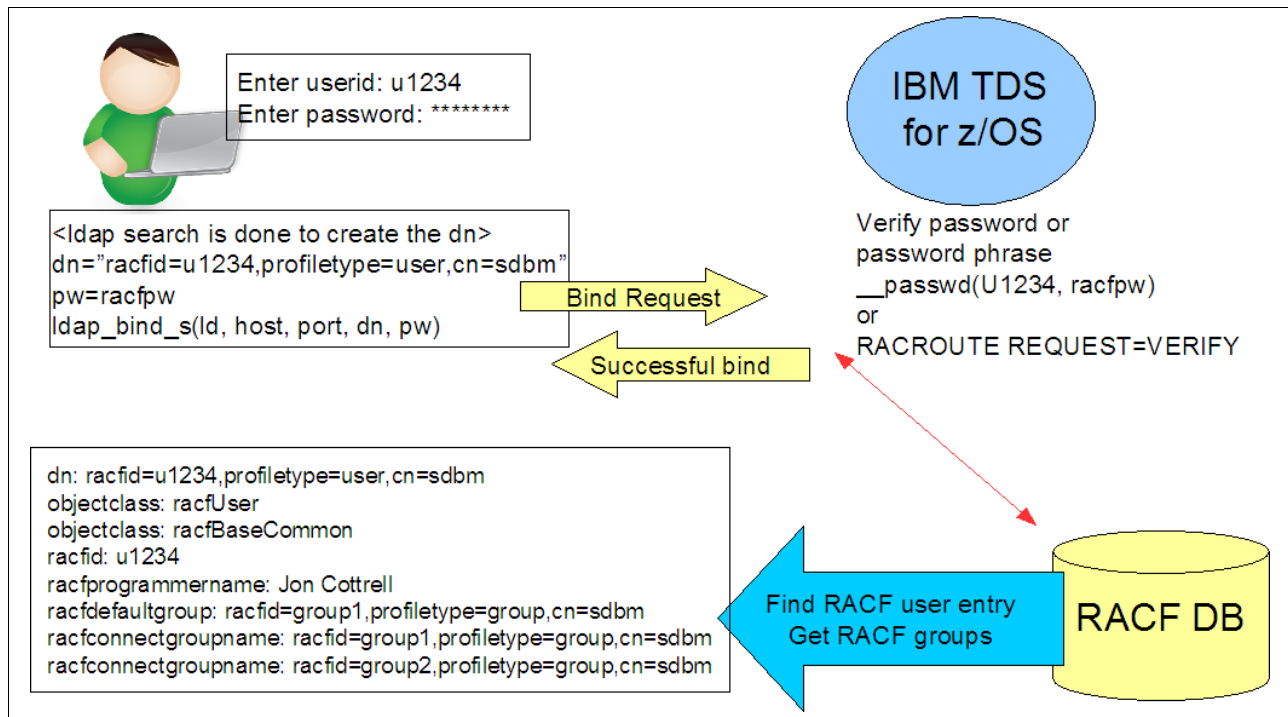


Figure 5-6 SDBM simple bind example

A disadvantage of storing passwords in TDBM, LDBM, or CDBM back end entries is that this results in managing yet another password repository if there are already users and passwords setup in the z/OS Security Manager (e.g. RACF). Also, there is the concern about protecting these password (**userPassword**) values in the entries with the use of ACLs or password encryption or hashing.

There are several disadvantages of performing simple binds with the SDBM back end:

- ▶ A long distinguished name (DN) is required that uses nonstandard LDAP server schema. For example instead of using standard LDAP attribute types and values such as `cn`, `o`, and `c`; an SDBM DN uses `racfid` and `profiletype` attribute values. Most LDAP administrators are familiar with standard LDAP schema and might not be as familiar with the schema that is used by IBM Tivoli Directory Server for z/OS for the SDBM back end.
- ▶ The SDBM back end has limited search capabilities.
- ▶ Unless there are many additional RACF custom fields set up, it is not easy to add additional data to RACF users.

To overcome these limitations of performing simple binds to TDBM, LDBM, CDBM, and SDBM back ends, native authentication support was added to IBM Tivoli Directory Server for z/OS. This support allows the usage of the entries that reside in the TDBM, LDBM, or CDBM back ends, but the passwords or password phrases are actually stored in the z/OS Security Server. Thus, the Security Server performs the password verification on simple bind requests. Because the Security Server stores the password, there is no need to manage or synchronize another password repository. Also because the entry data is stored in the TDBM, LDBM, or CDBM back end, additional attribute types and values (data) can easily be added to entries.

Figure 5-7 illustrates an LDAP client application performing a simple bind to an entry residing in the TDBM, LDBM, or CDBM back end that participates in native authentication. First the application does an LDAP search with a base distinguished name (DN) of `o=ibm,c=us` and a filter of `cn=jonc` to find the entry in the directory. If the entry is found, the LDAP client application then performs a simple bind or authentication as that entry participating in native authentication by specifying a password of `racfpw`. The `racfpw` value is specified on the simple bind request because that is the password value for the RACF user, `u1234`.

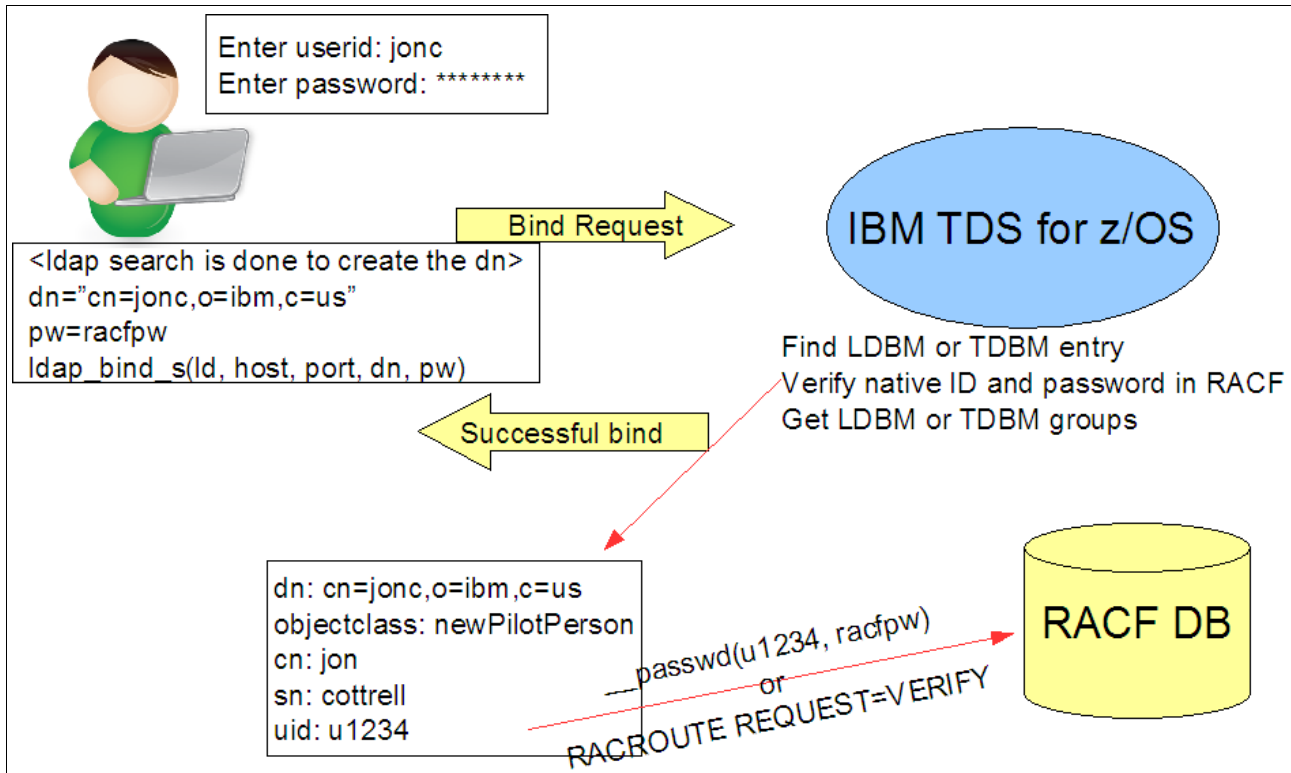


Figure 5-7 TDBM and LDBM native authentication bind example

5.3.1 Setting up native authentication

The LDBM, TDBM, or CDBM back end entries must specify either a single `uid` or `ibm-nativeId` attribute value that specifies the user ID in the Security Server (e.g. RACF) that is to be used when performing the simple bind request.

Note the following:

1. If using the `ibm-nativeId` attribute type to store the user ID, the LDBM, TDBM, or CDBM entries must be updated to add an objectclass value of `ibm-nativeAuthentication` and an `ibm-nativeId` value pointing to a valid Security Server user ID. The `ibm-nativeId` attribute type and `ibm-nativeAuthentication` objectclass are in the minimum or initial schema.
2. If the `uid` attribute type is used to store the user ID, the object class value on the LDBM, TDBM, or CDBM entry must allow the specification of the `uid` attribute value and only one `uid` attribute value can be specified. The `uid` attribute type is multi-valued in the LDAP server's schema.
3. The user that is specified in the `uid` or `ibm-nativeId` attribute value must have an OMVS segment defined with valid UID value specified in the z/OS Security Server.

After a decision has been made on how to store the user ID in the LDBM, TDBM, or CDBM entries, there is a set of native authentication options in the LDBM, TDBM, and CDBM back end sections of the LDAP server configuration file that must be updated:

- ▶ **useNativeAuth** [all | selected | off]
- ▶ **nativeAuthSubtree** [all | dn]
- ▶ **nativeUpdateAllowed** [on | off | reset]

The **useNativeAuth** configuration option controls the entries that are eligible for native authentication. When set to all, entries that contain the **uid** or **ibm-nativeId** attribute value are eligible for native authentication. When set to selected, only entries that contain an **ibm-nativeId** attribute value are eligible for native authentication. Of course, **off** means that native authentication is not active in the back end.

The **nativeAuthSubtree** multi-valued configuration option indicates which subtree(s) in the back end are eligible for native authentication. If this option is set to **all**, every entry in the back end is eligible for native authentication. Otherwise, this option can be specified multiple times in each back end to indicate which subtrees are eligible for native authentication.

The **nativeUpdateAllowed** configuration option controls whether passwords or password phrases can be changed on **modify** or **update** requests in the Security Server. When set to **on**, password or password phrases are allowed to be updated (as long as they are not already expired). When set to **off**, password or password phrases are not allowed to be updated on modify requests. When set to **reset** and the PasswordPolicy control (OID 1.3.6.1.4.1.42.2.27.8.5.1) is specified on the bind request, the password or password phrase can be modified even if the user's password or password phrase has already expired.

Note: The **nativeUpdateAllowed** reset value is only supported in z/OS V1R12 and later.

5.3.2 Changing a password or password phrase of an entry participating in native authentication

There are two ways of changing or updating a user's password or password phrase:

1. During a simple bind or authentication attempt
2. Performing a **modify-delete** and **modify-add** of the **userPassword** attribute value

To change the password or password phrase during the simple bind, specify **currentPassword/newPassword** as the password or password phrase value on the bind request. If the **currentPassword** is correct and the **newPassword** value is acceptable to the z/OS Security Server, the password is changed. In Figure 5-7 on page 106, the **cn=jon,o=ibm,c=us** entry uses the z/OS Security Server user ID, **u1234**, which currently has a password value of **racfpw**. This example uses the z/OS **ldapsearch** utility to change it from **racfpw** to **racf1pw** on a bind request.

```
ldapsearch -D "cn=jon,o=ibm,c=us" -w racfpw/racf1pw -s base -b "" "objectclass=*" 
```

Note: The **nativeUpdateAllowed** configuration option setting does not affect the ability to change the native password or password phrase on a simple bind request.

If **nativeUpdateAllowed on** is specified, the native password or password phrase is allowed to be changed on a modify operation (assuming that the bound user is authorized to do so in the Security Server). This is done by modifying the LDBM, TDBM, or CDBM entry to delete the current password value and then adding the new password value. To accomplish this

password or password phrase change, a special type of **modify** (**modify-delete** followed by **modify-add**) involving the **userPassword** attribute value must be done.

For example, `cn=jon,o=ibm,c=us` uses z/OS Security Server user ID, `u1234`, which now has a password value of `racf1pw`. This example uses the z/OS **1dapmodify** utility to change it to `racf2pw`.

```
1dapmodify -D "cn=jon,o=ibm,c=us" -w racf1pw -f racfpw.ldif
```

Example 5-1 shows the steps in the change.

Example 5-1 Changing password for user ID u1234 from racf1pw to racf2pw

```
cn=jon,o=ibm,c=us  
-userpassword=racf1pw  
+userpassword=racf2pw
```

Note the following:

1. The **userPassword** attribute is used as a mechanism to change the native password or password phrase, but an entry that is using native authentication cannot actually include the **userPassword** attribute.
2. The z/OS **1dapchangepwd** utility shipped in z/OS V1R12 and later can be used in place of the **1dapmodify** utility to change the password or password phrase. See *IBM Tivoli Directory Server Client Programming for z/OS V1R12.0*, SA23-2214-04 for more information about the **1dapchangepwd** utility.

For more information about native authentication, see *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

5.4 Authorization using Tivoli Directory Server Access Control Lists (ACL)

Generally speaking, there is a set of actions that a subject must be authorized to perform on objects in an LDAP directory. As previously defined, the objects to secure are directory entries. Directory entries consist of attributes with one or more values. Most often a subject seeks authorization to perform an action against one or more attribute-value pairs within an entry. These attribute level actions are: **read**, **write**, **search**, and **compare**.

The subject seeking authorization is identified by its bind credentials, most notably its bind DN (but could include other DNs depending on the bind mechanism). In addition to its bind credentials, the subject can belong to groups. Group membership facilitates the authorizing of a common set of actions for groups of subjects.

Similar to grouping subjects, the attributes within an entry belong to one of five attribute classes:

- ▶ **normal**
- ▶ **sensitive**
- ▶ **critical**
- ▶ **system**
- ▶ **restricted**

An attribute's class is defined in the schema. See Chapter 4, “Schemas” on page 73 for more details. This allows an LDAP administrator to authorize on an attribute-class-basis as opposed to explicitly authorizing every attribute in the entry.

Given the nature of most directories, attribute level authorization is more common. Consider an employee directory. In most cases, the number of employees will remain fairly constant. However, subjects might need authorization to add or delete objects. As a result, an administrator can authorize subjects to **add** or **delete** entries. Authorization to delete allows a subject to delete an LDAP entry, whereas add authorization allows a subject to add a direct child entry.

Finally, to facilitate an LDAP administrator, IBM Tivoli Directory Server supports the concept of entry ownership. If a subject is granted entry ownership, they will have full authority over the object. That is, attribute **read**, **write**, **search**, and **compare**, and object **add** and **delete** actions will be authorized for the owner.

In the upcoming sections, we answer the following:

- ▶ How does an administrator authorize a subject in IBM Tivoli Directory Server?
- ▶ Who exactly is the subject seeking access?
- ▶ How does an administrator consider more than a subject's identification, such as their location or time of day, when authorizing?
- ▶ How can an administrator be sure the configuration set in LDAP satisfies all security requirements?

Note: This section covers securing LDAP entries stored in IBM Tivoli Directory Server. RACF authorization is not part of this chapter. RACF-style bind DN's apply here, but READ, UPDATE, ALTER, CONTROL authority on RACF profiles will not be covered.

5.4.1 Setting up IBM Tivoli Directory Server Authorization

IBM Tivoli Directory Server for z/OS authorization is based on ACLs defined for entries within the directory. The ACLs are defined using basic entry attributes defined in the schema. In addition, entry owners can also be defined using entry attributes. The following table introduces all of the attributes related to ACLs and entry owners. Some are set by the LDAP administrator, whereas others are informational attributes managed by IBM Tivoli Directory Server and not modifiable by an LDAP administrator. Recall all LDAP attributes have an associated access class. Table 5-5 also lists the access class for the authorization related attributes.

Table 5-5 access class for the authorization related attributes

Attribute	Access Class	Syntax
aclEntry	Restricted	Multi-valued Directory String
entryOwner	Restricted	Multi-valued Directory String
aclPropagate	Restricted	Single value Boolean
ownerPropagate	Restricted	Single value Boolean
aclSource	System	Single value DN (Calculated value by IBM Tivoli Directory Server)
ownerSource	System	Single value DN (Calculated value by IBM Tivoli Directory Server)

When setting up authorization, an LDAP administrator assigns **aclEntry** and **entryOwner** values to entries. The diagrams in Figure 5-8 and Figure 5-9 illustrate the syntax of both attributes:

```

aclEntry: aclEntry_value

aclEntry_value :- [access-id:|group:|role:]subject_DN[granted_rights]
or,
aclEntry_value :- aclFilter:filter:operation[granted_rights]
  - subject_DN :- valid DN, the subject that privileges are authorized or denied.
  - filter :- valid search filter, using the following attributes only:
    • ibm-filterSubject
    • ibm-filterIP
    • ibm-filterTimeOfDay
    • ibm-filterDayOfWeek
    • ibm-filterBindMechanism
    • ibm-filterConnectionEncrypted.
  - operation :- union | replace | intersect.
  - attr_rights :- at.attr_name:|[grant:|deny:]|attr_rights_list
    • attr_name :- any valid attribute name
    • attr_rights_list :- [r|w|s|c]
  - granted_rights :- object_rights | normal_rights | sensitive_rights | critical_rights |
restricted_rights | system_rights | attr_rights
    • object_rights :- object:|[grant:|deny:]|object_rights_list
    • object_rights_list :- [a|d]
    • normal_rights :- normal:|[grant:|deny:]|attr_rights_list
    • sensitive_rights :- sensitive:|[grant:|deny:]|attr_rights_list
    • critical_rights :- critical:|[grant:|deny:]|attr_rights_list
    • restricted_rights :- restricted:|[grant:|deny:]|attr_rights_list
    • system_rights :- system:|[grant:|deny:]|attr_rights_list
    • attr_rights_list :- [r|w|s|c]

```

Figure 5-8 *aclEntry* syntax

```

entryOwner: entryOwner_value

entryOwner_value :- [access-id:|group:|role:]subject_DN
or,
entryOwner_value :- ownerFilter:filter:grant|deny
  - subject_DN :- valid DN, represents the subject that privileges are granted/denied.
  - filter :- valid search filter, using the following attributes only:
    • ibm-filterSubject
    • ibm-filterIP
    • ibm-filterTimeOfDay
    • ibm-filterDayOfWeek
    • ibm-filterBindMechanism
    • ibm-filterConnectionEncrypted.

```

Figure 5-9 *entryOwner* syntax

Every entry in the directory must have an **ac1Entry** and **entryOwner** associated with it. If no **ac1Entry** and **entryOwner** is defined in the directory explicitly, a default is set by IBM Tivoli Directory Server:

```
ac1Entry: group:CN=ANYBODY:normal:rsc:system:rsc
entryOwner: administratorDN (DN set in ds.conf for administrator)
```

On IBM Tivoli Directory Server for z/OS, for performance reasons, if no **ac1Entry** and **entryOwner** values are defined, all suffix entries added to the directory are updated to include these values.

5.4.2 Normalization

One key LDAP standard requirement is that all values for multi-valued attributes within an entry are unique. To enforce uniqueness, IBM Tivoli Directory Server relies on normalization rules for each attribute. **entryOwner** and **ac1Entry** attributes have distinct normalization rules. In the case of non-filtered **entryOwner** and **ac1Entry** attributes, the normalized value is the normalized subject_DN field from Figure 5-8 on page 110 and Figure 5-9 on page 110.

On the other hand, in the case of filtered ACLs, the normalized filter field from the syntax diagrams is used as the normalized value. The normalization of a filter consists of normalizing all predicts of the filter. The table lists the normalization rules and other restrictions for the supported attributes within a filter used for filtered ACLs.

Table 5-6 Normalization rules

Attribute	Normalization	Other Filter Requirements
ibm-filterBindMechanism	Uppercase	Substring not supported in filter. Value can be one of the following only: GSSAPI EXTERNAL SIMPLE CRAM-MD5 DIGEST-MD5
ibm-filterConnectionEncrypted	Boolean normalization	
ibm-filterDayOfWeek	Integer normalization	Values can only be from 0 - 6, where: 0=Sunday ... 6=Saturday. All others are rejected.
ibm-filterTimeOfDay	None	Values can only be of the format: HH:MM. All others are rejected. Substring not supported in filter.
ibm-filterSubject	DN normalization	Substring matching supported in filter

Attribute	Normalization	Other Filter Requirements
ibm-filterIP	<p>All IPV4 octets are expanded to 3 digits</p> <p>All IPV6 octets are expanded to 4 digit hex</p> <p>Double colons in IPV6 are expanded</p>	<p>IPv4/IPv6 values supported.</p> <p>Substring matching supported only when wildcard is specified as the last digit:</p> <p>9.12.5* 9.12.134.* 2001:0db8:85a3:08d3:1319:8a2e:0370:73*</p>

5.4.3 Propagation

To facilitate authorization administration, IBM Tivoli Directory Server supports the concept of propagating ACLs and entry ownership. As previously stated, LDAP directories consist of entries organized in a hierarchical tree layout. That is, parent entries have descendant entries, which in turn can be parents to other entries. Given this layout, propagation allows for an administrator to set ACLs and entry owners with limited number of updates. This is accomplished by setting **ac1Entry** and **entryOwner** values in conjunction with TRUE values for the **ac1Propogate** and **ownerPropogate** attributes on parent entries. This results in the **ac1Entry** and **entryOwner** values propagating to any descendant entries. These values propagate down the tree until a descendant entry defines its own **ac1Entry** and **entryOwner** values. If this descendant entry sets the propagate values to TRUE, then its values propagate to all of its descendants. On the other hand, if the descendant does not propagate its values, its descendant children entries' **ac1Entry** and **entryOwner** values are defined by its nearest parent that is propagating. See Figure 5-10.

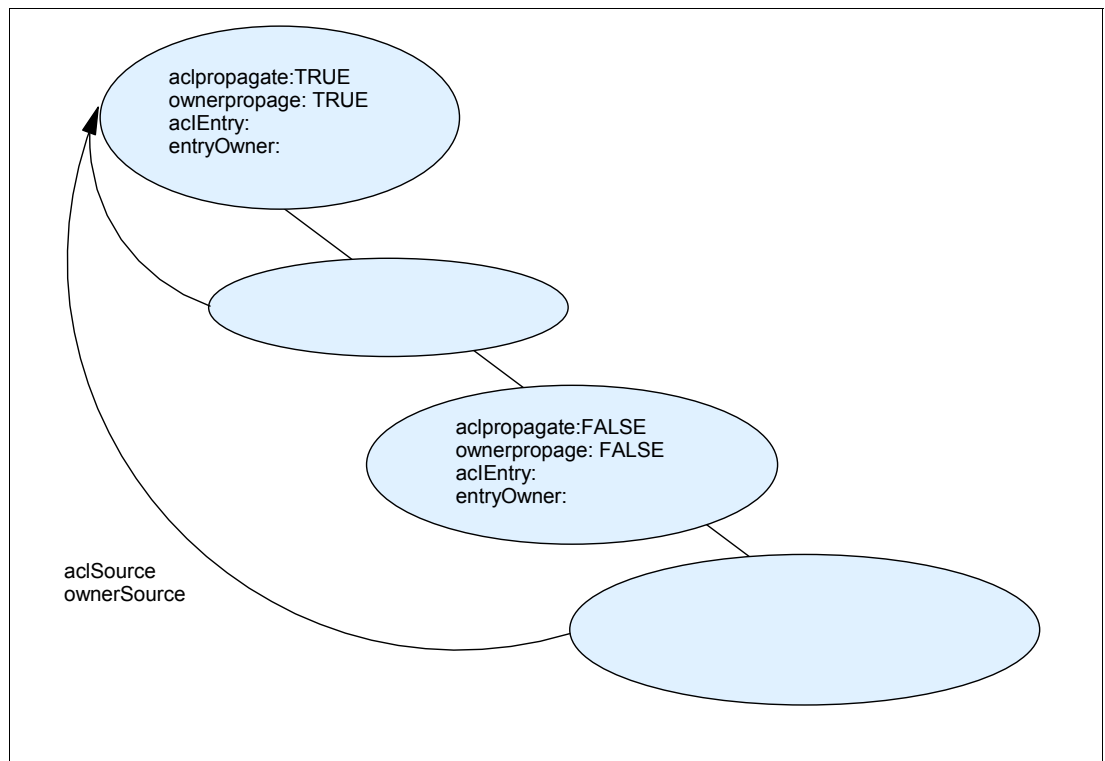


Figure 5-10 Propagation

Clearly, the concept of propagation allows for minimal updates. However, given the possibility of large/complex directory hierarchies with many authorization requirements, things can get confusing for an administrator. The **ac1Source** and **ownerSource** attributes help. They are attributes that can be returned on an LDAP search (assuming the client has authority). Their values are the DNs of the entry that defines the **ac1Entry** and **entryOwner** values for the entry returned on the search.

As we will see in upcoming sections, propagation is just one advanced concept related to authorization. New to IBM Tivoli Directory Server for z/OS R12 is the concept of filtered access control. Filtered access control provides great flexibility in setting up authorization within IBM Tivoli Directory Server. Needless to say, with flexibility comes opportunity for error. To provide administrators a means of verification of their authorization configuration, IBM Tivoli Directory Server for z/OS R12 provides an extended operation to retrieve the **ac1Entry** and **entryOwner** values for entries and their implications to client access. Before discussing the extended operation, we must discuss in detail what exactly **ac1Entry** and **entryOwner** values mean and define the concept of effective Access-Class/Attribute/Object Permissions.

5.4.4 Authorization Permissions

As described in the preceding sections and depicted in the **ac1Entry** value syntax, IBM Tivoli Directory Server authorization is based on six types of permissions. The following four permissions are available for attributes, either using the access class, the specific attribute, or both:

- ▶ **read**
- ▶ **write**
- ▶ **search**
- ▶ **compare**

For objects, two permissions are available:

- ▶ **add**
- ▶ **delete**

The **ac1Entry** syntax also allows for explicit grant and deny statements to be included with the permissions. **grant** is optional/assumed when **attribute**, **object**, and **access-class** permissions are added to the **ac1Entry** value. Absence of an attribute or access-class in an **ac1Entry** value results in all permissions being denied for that attribute or all attributes for the access-class. Absence of the object permissions in an **ac1Entry** value results in object permissions being denied.

When multiple **ac1Entry** values apply to a subject, the permissions are combined. Later we will discuss the subject and **ac1Entry/entryOwner** matching rules. The final results of combining all applicable **ac1Entry** values are known as the effective access class permissions, effective object permissions, and effective attribute permissions. Note **deny** takes precedence when performing calculation. The following shows an example of basic effective access class, object, and attribute permissions calculation for a subject that belongs to group1 and group2. Deny processing is shown with **at.sn**.

```
ac1entry: group: cn=group1:normal:rw:at.cn:rWSC: at.sn: deny:w:object:a
ac1entry: group: cn=group2:system:sw:at.sn:rWSC:object:ad
```

The results of these commands are:

```
system:sw:normal:rw
at.cn:rWSC:at.sn:rsc: OR at.cn:rWSC:at.sn:grant:rsc:deny:w
object:ad
```

5.4.5 Precedence

In the example above we show the precedence of **deny** over **grant** when combining permissions for the same attribute. There is also a set of rules when attribute level and access-class permissions for the same attribute must be combined. The following summarizes the complete set of precedence rules IBM Tivoli Directory Server follows when performing authorization checking. It factors in entry ownership.

1. Subject is entry owner → **grant**
2. Attribute level deny → **deny**
3. Access class level deny → **deny**
4. Attribute level grant → **grant**
5. Access class level grant → **grant**
6. No effective access class or effective attribute level permissions for an attribute → **deny**

A combination of permissions and following a set of precedence rules is the most straightforward aspect of IBM Tivoli Directory Server's authorization engine. Subject determination is not as straightforward.

5.4.6 Determining the Subject

In IBM Tivoli Directory Server, the subject seeking authorization is represented by one or more distinguished names. Table 5-7 describes all of them.

Table 5-7 Subject distinguished name

DN	Notes
Basic Bind DN	Includes RACF style DNs. Specified on bind by the client. If using SASL EXTERNAL bind, it is the subject in X.509 certificate. If using Kerberos, the Kerberos style DN also becomes the basic Bind DN.
Kerberos style DN	Basic Bind DN generated by IBM Tivoli Directory Server when Kerberos binds are done. The Kerberos principal name is used to form the DN. <i>ibm-kn= principalName@krbRealm</i>
1-n Alternate DNs	Alternate DNs are associated with RACF mapped certificates in SASL EXTERNAL binds and Kerberos mapped IDs.
Special IBM Tivoli Directory Server DN: cn=this	This DN can be used when a client is working with the entry representing it stored in IBM Tivoli Directory Server.
1-m Group DNs	DNs can be added to group objects in IBM Tivoli Directory Server. These DNs are a list of all group objects the bind DN and alternate DNs belong to.
Special IBM Tivoli Directory Server DN: cn=authenticated	This DN can be used when working with non-anonymous clients.
cn=anybody	All clients belong to this group. Used commonly to set authorization for anonymous clients.

Depending on the bind mechanism and group membership, one of the DNs (or in the case of alternate and group DNs, the group DN or alternate DN set) shown in the table are compared against the subject_DN segment within **aciEntry/entryOwner** values. If using filtered access

control, in addition to the subject DN, more details are used to compare against the filter segment of **ac1Entry/entryOwner** values. For all that match, the values are included in the calculation of the effective attribute, access-class, and object permissions. Similar to the preceding rules for effective permissions calculation, the client's possible subject DN(s) are checked in a specific order. After a DN from the table matches, it will serve as the subject for the rest of the process. This is especially important when it comes to filtered ACLs. In the next section we put everything together to describe the IBM Tivoli Directory Server authorization engine in detail.

5.4.7 Calculating Effective Permissions

The following describes all of the basic steps followed with the IBM Tivoli Directory Server authorization engine. As described above, the **entryOwner** and **ac1Entry** attributes are associated with the LDAP entry being accessed by a client on a given request.

Also, prior to describing the steps, it is important for the reader to understand that permissions calculation is done using basic set arithmetic, i.e., **union**, **intersect**, and **replace**. The permissions granted or denied can be seen as elements within a basic set.

1. Check basic bind DN against all **entryOwner** values.
If an **entryOwner** value matches, grant all permissions and EXIT authorization processing.
If no matches, go to next step.
2. Check all alternate DNs against all **entryOwner** values.
If an **entryOwner** value matches for any of the alternate DNs, grant all permissions and EXIT authorization processing.
If no matches, go to next step.
3. Check all group DNs against all **entryOwner** values.
If an **entryOwner** value matches for any of the group DNs, grant all permissions and EXIT authorization processing.
If no matches, go to next step.
4. Check basic bind DN against all **ac1Entry** values.
For all **ac1Entry** values that match, union all the permissions within the matching values to perform the effective permissions calculation.
Perform authorization check using the calculated effective permissions and considering access-class versus attribute level precedence rules.
If no **ac1Entry** values match, go to next step.
5. Check all alternate DNs against all **ac1Entry** values.
For all **ac1Entry** values that match, union all the permissions within the matching values to perform the effective permissions calculation.
Perform authorization check using the calculated effective permissions and considering access-class versus attribute level precedence rules.
If no **ac1Entry** values match, go to next step.
6. If the basic bind DN matches the entry being accessed, check the **cn=this** DN against all **ac1Entry** values.
For all **ac1Entry** values that match, union all the permissions within the matching values to perform the effective permissions calculation.
Perform authorization check using the calculated effective permissions and considering access-class versus attribute level precedence rules.
If no **ac1Entry** values match or basic bind DN does not equal the entry being accessed, go to next step.
7. Check all groups DNs against all **ac1Entry** values.
For all **ac1Entry** values that match, union all the permissions within the matching values to perform the effective permissions calculation.
Perform authorization check using the calculated effective permissions and considering

access-class versus attribute level precedence rules.

If no **ac1Entry** values match, go to next step.

8. If the subject is not anonymous, that is, a successful LDAP bind was performed prior to access, check the **cn=authenticated** DN against all **ac1Entry** values.
For all **ac1Entry** values that match, union all the permissions within the matching values to perform the effective permissions calculation.
Perform authorization check using the calculated effective permissions and considering access-class versus attribute level precedence rules.
If no **ac1Entry** values match or anonymous client, go to next step.
9. Check the **cn=anybody** DN against all **ac1Entry** values.
For all **ac1Entry** values that match, union all the permissions within the matching values to perform the effective permissions calculation.
Perform authorization check using the calculated effective permissions and considering access-class versus attribute level precedence rules.
If no **ac1Entry** values match, no permissions are granted and thus authorization is denied.

Filtered access control extends this process.

5.4.8 Filtered Access Control

IBM Tivoli Directory Server's standard authorization engine can be summarized in the following steps:

1. Determine all the possible Subject DN's for a given client.
2. For the LDAP entry being accessed, determine all of its **ac1Entry** and **entrOwner** values, using propagation rules if necessary.
3. Compare values from step 2 against Subject DN's from step 1, gathering all that match (following subject matching order rules).
4. Calculate effective permissions from permissions gathered in step 3 by performing the union operation on all matching values.
5. Follow precedence rules and perform authorization check using the results from Step 4.

For IBM Tivoli Directory Server for z/OS R12, the authorization engine was extended to include the concept of filtered access control. Essentially filtered access control extends step 3 from the above list. In addition to matching the Subject DN's, other client details are used to compare against **ac1Entry** and **entryOwner** values that use the new filtered syntax. Step 4 also is extended because the calculation of the effective permissions is given more flexibility.

The updated list of operations for the engine now follows:

1. Determine all the possible Subject DN's for a given client.
2. For the LDAP entry being accessed, determine all of its **ac1Entry** and **entrOwner** values, using propagation rules if necessary.
3. Compare non-filtered values from step 2 against Subject DN's from step 1, gathering all that match following subject matching order rules.
4. Perform the union operation on the results of step 3 while following precedence rules that specifically handle the **deny** operator.
5. If filtered values exist, determine the client's values for the following:
 - a. IP
 - b. Bind Mechanism
 - c. Subject DN -- This is set by the DN or DN's that match step 3. If none match in step 3, the order of the DN's used in step 3 is followed until a filtered value matches.
 - d. Encrypted connection True or False
 - e. Time of access
 - f. Day of access

6. Using the data determined in step 5, compare against all filtered values. If filtered **entryOwner** values match and none have the deny operator, the client is considered the entry owner and given all permissions. The engine then returns. Otherwise, filtered **ac1Entry** values that match are gathered. The filtered **ac1Entry** value's operator is considered and the union operation is performed to form 3 separate sets for values with the like operators:
 - a. Replace Set
 - b. Union Set
 - c. Intersect Set
7. Calculate effective permissions from permissions gathered in step 4 and 6.
 - a. If replace set found, discard results of Step 4.
 - b. If union set found, union permissions in the union set with the results in Step 6.a or results from step 4 if no replace set is found.
 - c. If intersect set found, intersect permissions in the intersect set with the results in Step 7.b, results from Step 7.a if union set not found, or Step 4 if no union set and no replace set found. The intersect operation is not performed with any deny permissions. Instead, any deny permissions found in any of the three sets will be an element in the final effective permissions set.
8. Perform authorization check against results from Step 7.

A note about Administrator

Prior to filtered access control, the Administrator defined in the `ds.conf` file had all permissions granted. Now filtered access control allows for an administrator whose client information, such as IP, time of access, or day of access, matches filtered **ac1Entry** values have a reduced set of permissions.

5.4.9 Testing Authorization Configurations

Throughout this section we described the details of the IBM Tivoli Directory Server authorization engine. The authorization engine requires configuration by an LDAP administrator. Because configuration is done using LDAP entries, attributes, and other concepts like search filters, an administrator is afforded a considerable amount of flexibility. Clearly, this flexibility is only beneficial if all organizational security requirements are met. With large LDAP directories and complex requirements addressed by the filtered access control feature, administrators must have a means of verifying the configuration.

As mentioned earlier, the **ac1Source** and **ownerSource** attributes offer a basic indication of how **ac1Entry** and **entryOwner** values will work. These attributes allow for a quick verification of the propagation settings.

To provide a more complete picture of the configuration, IBM Tivoli Directory Server for z/OS R12 provides an extended operation: **GetEffectiveAc1**. This extended operation, issued using the **1dapexop** utility from a UNIX Systems Services shell, allows an administrator to check a client's effective access-class, attribute, and object permissions for any LDAP entry. Among other details, it takes as options all of the parameters required by filtered access control. This allows an administrator to simulate a client request and test the effective permissions that will be used by the authorization engine. The command syntax is shown in Figure 5-11 on page 118. Note that we only highlight the syntax for **GetEffectiveACL**. **1dapexop** has general options that are used in addition to the options listed below. Refer to *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 entire **1dapexop** tool syntax.

```

-op geteffectiveacl -filt searchFilter [-b baseDN] [-s {base | one | sub}] [-a
{never | always | search | find}] [-z sizeLimit] [-l timeLimit]

[-m {SIMPLE | CRAM-MD5 | DIGEST-MD5 | GSSAPI | EXTERNAL}] [-dn DN] [-u
racfUserID] [-r principalRealm] [-ip bindIP] [-time accesTime] [-day
accessDay] [-en]

```

Figure 5-11 *geteffectiveacl* partial syntax

The syntax can be seen as defining two components of the command (hence the two lines):

1. Basic LDAP Search options to retrieve the objects being secured by the authorization engine
2. Client information needed to calculate the effective permissions on the objects retrieved

Earlier in this section we listed the additional client information that can be taken into account when calculating the effective permissions. Table 5-8 summarizes this information with the extended operation option and the LDAP attribute used within a filtered **aclEntry** or **entryOwner** value.

Table 5-8 *Extended operation*

Client detail	LDAP attribute	Extended Operation option(s)
IP	ibm-filterIP	-ip
Subject	ibm-filterSubject	-dn -r -u Note groups and alternate DNs are determined by using these options, the bind mechanism option (-m), the groups in the directory, and DN mapping rules setup in IBM Tivoli Directory Server.
Bind Mechanism	ibm-filterBindMechanism	-m Note this value can impact the subject used.
Connection Encrypted	ibm-filterConnectionEncrypted	-en
Day of Access	ibm-filterDayOfWeek	-day
Time of Access	ibm-filterTimeOfDay	-time

GetEffectiveAcl Example

The following shows an example of the **getEffectiveAcl** extended operation.

In our example, we use **ldapexop** to perform a subtree search to return two objects. The client is a member of two groups and has alternate DNs associated with it. Note the **aclEntry** and **entryOwner** source information is also presented.

```
ldapexop -D adminDn -w adminPw -op geteffectiveacl -filter "objectclass=*"
-base "dc=yourcompany,dc=com" -s sub -a never -z 100 -l 10
-dn "cn=Joe Shmoe,ou=users,dc=yourcompany,dc=com" -ip 129.176.132.92
-time 18:30 -day 6 -mech SIMPLE -encrypt
```

Figure 5-12 *ldapexop search command*

```
#ENTRY INFORMATION:
dn: dc=yourcompany,dc=com

#SUBJECT INFORMATION:
#Bind DN:
dn: cn=Joe Shmoe,ou=users,dc=yourcompany,dc=com

#Alternate DNs:
dn: cn=alt_01
dn: cn=alt_02

#Group DNs:
dn: cn=group_01
dn: cn=group_02

#SOURCE ATTRIBUTE VALUES:
aclEntry: group:cn=Anybody:normal:rsc:system:rsc
aclPropagate: TRUE
aclSource: dc=yourcompany,dc=com
entryOwner: cn=Admin
ownerPropagate: TRUE
ownerSource:dc=yourcompany,dc=com

#APPLICABLE ATTRIBUTE VALUES:
aclEntry: group:cn=Anybody:normal:rsc:system:rsc

#EFFECTIVE ACCESS-CLASS PERMISSIONS:
normal: grant:rsc
system: grant:rsc
```

Figure 5-13 *ldapexop search command results - first object*

```

#ENTRY INFORMATION:
dn: ou=users,dc=yourcompany,dc=com

#SUBJECT INFORMATION:
#Bind DN:
dn: cn=Joe Shmoe,ou=users,dc=yourcompany,dc=com

#Alternate DNs:
dn: cn=alt_01
dn: cn=alt_02

#SOURCE ATTRIBUTE VALUES:
aclEntry: group:cn=Anybody:normal:rsc:system:rsc
aclPropagate: TRUE
aclSource: dc=yourcompany,dc=com
entryOwner: cn=Joe Shmoe,ou=users,dc=yourcompany,dc=com
ownerPropagate: TRUE
ownerSource: dc=yourcompany,dc=com

#APPLICABLE ATTRIBUTE VALUES:
entryOwner: cn=Joe Shmoe,ou=users,dc=yourcompany,dc=com

#EFFECTIVE ACCESS-CLASS PERMISSIONS:
restricted:grant:rwc
system:grant:rwc
critical:grant:rwc
sensitive:grant:rwc
normal:grant:rwc
object:grant:ad

```

Figure 5-14 Idapexop results - second object

5.4.10 Closing thoughts on authorization

IBM Tivoli Directory Server for z/OS's authorization engine provides administrators flexibility when securing the directory. Because its initial release, concepts like groups, attribute access classes, and propagation lessen the number of updates to the directory needed to configure the authorization engine. In R12, IBM Tivoli Directory Server for z/OS's authorization engine was extended. In addition to static client information like the bind DN, more dynamic client information can now be used to enhance security. For organizations that are bound by laws or regulations requiring tighter, more dynamic control, information like the client's location or time of access might have security implications. Filtered access control allows for an administrator to address these concerns. Of course all of this flexibility can lead to confusion or error when performing administration. IBM Tivoli Directory Server for z/OS addresses this with the **GetEffectiveAcl** extended operation. When planning, administrators should consider the following:

- ▶ ACLs governing a client's entry within the directory most often should allow, at minimum, the client write access to the **userpassword** attribute. Otherwise, all password modifications by the client would require an administrator. Considering the password policy feature, this should be avoided.
- ▶ Access class permissions and propagation should be used to reduce the number of **aclEntry** and **entryOwner** values required in the directory.

- ▶ Group membership should be used to reduce the number of **ac1Entry** and **entryOwner** values required in the directory. When considering groups, one must consider the choice of dynamic groups versus static groups.
- ▶ When using filtered access control, the union, intersect, and replace operators can be leveraged to reduce the number of **ac1Entry** values required in the directory. Instead of trying to have an **ac1Entry** value for every possible situation dictated by the client's dynamic information, an incremental approach can be used such that all cases can be covered with a minimal number of values.
- ▶ The **GetEffectiveAc1** extended operation should be leveraged to verify the configuration, even if not using filtered access control.

5.5 Groups and group gathering in IBM Tivoli Directory Server for z/OS

To help simplify authorizing users to entries in the LDBM, TDBM, and CDBM back ends, an LDAP administrator can place similar users (such as those in the same department or project) into static, dynamic, or nested groups, and then authorize those groups to the desired resources in the directory. See 5.4, "Authorization using Tivoli Directory Server Access Control Lists (ACL)" on page 108 for more information about authorization.

This section discusses:

1. The supported LDAP static, dynamic, and nested groups
2. How to query group membership
3. The pros and cons of static, dynamic, and nested groups
4. How group gathering works after successfully authenticating to the LDAP server

5.5.1 Static, dynamic, and nested groups

A static group is a group where membership is statically defined by an LDAP administrator by adding the user's distinguished name (DN) as a **member** or **uniqueMember** attribute value. When using the **member** attribute, the entry must have an object class value of **accessRole**, **accessGroup**, **groupOfNames**, or **ibm-staticGroup** to be considered a static group. When using the **uniqueMember** attribute, the entry must have an object class value of **groupOfUniqueNames**.

```
dn: cn=staticgroup,o=ibm,c=us
objectclass: accessRole
cn: staticgroup
member: cn=jon,o=ibm,c=us
member: cn=saheem,o=ibm,c=us
member: racfid=karen,profiletype=user,cn=racf
```

Figure 5-15 Static group example in LDIF format

As illustrated in Figure 5-15, the members of a static group can reside in other back ends. For example, the member **racfid=karen,profiletype=user,cn=racf** resides in the SDBM back end.

A dynamic group is a group whose membership is determined by evaluating an LDAP search expression. The LDAP search expression includes the search scope, base distinguished name or baseDN (place in the directory where the search starts from), and the search filter. The search filter is allowed to be as complex as necessary to exactly define your dynamic group. These search expressions are allowed to be placed on the multi-valued **memberURL**

attribute. To be considered a dynamic group entry, the entry must have an object class value of **groupOfURLs** or **ibm-dynamicGroup**.

The following simplified LDAP URL syntax, shown in Figure 5-16, must be used as the value of **memberURL** attribute to specify the dynamic group search expression.

```
ldap:///baseDN[??[searchScope][?searchFilter]]
```

where

baseDN

Specifies the DN of the entry from which the search begins in the directory. The dynamic URL is not used if the base entry is not within the same back end as the dynamic group entry. This parameter is required. See Figure 5-18 on page 122 for an example of an baseDN that is not in the same back end as the dynamic group entry.

searchScope

Specifies the extent of the search. The default scope is base.

base Returns information aboutly about the baseDN specified in the URL.

one Returns information about entries one level below the baseDN

specified in the URL. It does not include the baseDN.

sub Returns information about entries at all levels below and including the baseDN.

searchFilter

Is the filter that you want applied to the entries within the scope of the search. See `ldapsearch` in *IBM Tivoli Directory Server Client Programming for z/OS V1R12.0*, SA23-2214-04 for additional information about LDAP search filters. The default is "objectclass=*".

Figure 5-16 *memberURL* attribute format

Figure 5-17 illustrates that the members of the `cn=testers,o=ibm,c=us` is all entries under the `c=us` entry that have a title attribute value of `tester`.

```
dn: cn=testers,o=ibm,c=us
objectclass: groupOfURLs
cn: tester
memberURL: ldap:///c=us??sub?(title=tester)
```

Figure 5-17 *Dynamic group entry example in LDIF format*

Figure 5-18 illustrates an example where the baseDN is not in the same back end as the dynamic group entry. The baseDN `o=nothere` does not exist as a valid suffix in the back end where the dynamic group entry resides.

```
dn: cn=notvalid_dynamic_group,o=nothere
objectclass: groupOfURLs
cn: notvalid_dynamic_group
memberURL: ldap:///o=nothere??one?(objectclass=person)
```

Figure 5-18 *Dynamic group entry example in LDIF format (baseDN does not exist in this back end)*

A nested group is defined as a group that references static, dynamic, and even other nested group entries. Nested groups allow LDAP administrators to construct and display group hierarchies that describe both direct and indirect group memberships. The nested group entries are allowed to be placed on the multi-valued **ibm-memberGroup** attribute. To be considered a nested group entry, the abstract **ibm-nestedGroup** objectclass must be specified, as shown in Figure 5-19.

```
dn: cn=testers_and_programmers,o=ibm,c=us
objectclass: ibm-nestedGroup
objectclass: container
ibm-memberGroup: cn=testers,o=ibm,c=us
ibm-memberGroup: cn=programmers,o=ibm,c=us
```

Figure 5-19 Nested group entry example in LDIF format

5.5.2 Querying group membership

As you can imagine, it is possible to introduce complexity when defining these static, dynamic, or nested groups. To help ease the administrative headaches of managing these users and groups, an LDAP administrator can request the **ibm-allGroups** and **ibm-allMembers** operational attributes on an LDAP search request. An operational attribute is one that must be explicitly requested to be returned.

The **ibm-allGroups** attribute indicates the distinguished names of all groups that the user entry belongs to, whereas the **ibm-allMembers** attribute displays all members of a particular group.

Assume the entries contained in the LDIF file in Figure 5-20 are added to IBM Tivoli Directory Server for z/OS.

```
dn: c=us
objectclass: country
c: us

dn: o=ibm,c=us
objectclass: organization
o: ibm

dn: cn=staticgroup,o=ibm,c=us
objectclass: accessRole
cn: staticgroup
member: cn=jon,o=ibm,c=us
member: cn=saheem,o=ibm,c=us
member: racfid=karen,profiletype=user,cn=racf

dn: cn=jon,o=ibm,c=us
objectclass: inetOrgPerson
cn: jon
sn: cottrell
uid: cottrell
userpassword: secret
title: programmer

dn: cn=saheem,o=ibm,c=us
objectclass: inetOrgPerson
cn: saheem
sn: granados
uid: granados
userpassword: secret
title: programmer

dn: cn=diane,o=ibm,c=us
objectclass: inetOrgPerson
cn: diane
sn: lia
uid: lia
userpassword: secret
title: tester

dn: cn=jeff,o=ibm,c=us
objectclass: inetOrgPerson
cn: jeff
sn: smith
uid: smith
userpassword: secret
title: tester

dn: cn=testers,o=ibm,c=us
objectclass: groupOfURLs
cn: testers
memberURL: ldap:///c=us??sub?(title=tester)

dn: cn=programmers,o=ibm,c=us
objectclass: groupOfURLs
cn: programmers
memberURL: ldap:///o=ibm,c=us??one?(title=programmer)

dn: cn=testers_and_programmers,o=ibm,c=us
objectclass: ibm-nestedGroup
objectclass: container
ibm-memberGroup: cn=testers,o=ibm,c=us
ibm-memberGroup: cn=programmers,o=ibm,c=us
```

Figure 5-20 Example static, dynamic, and nested group entries in LDIF format

Figure 5-21 shows the results of using the z/OS ldapsearch utility to query the **ibm-allMembers** and **ibm-allgroups** operational attribute types on an LDAP search.

```
ldapsearch -L -D "cn=admin" -w secret -s one -b "o=ibm,c=us" "objectclass=*"
ibm-allgroups ibm-allmembers
```

```
dn: cn=saheem,o=ibm,c=us
ibm-allgroups: cn=staticgroup,o=ibm,c=us
ibm-allgroups: cn=programmers,o=ibm,c=us
ibm-allgroups: cn=testers_and_programmers,o=ibm,c=us

dn: cn=jeff,o=ibm,c=us
ibm-allgroups: cn=testers,o=ibm,c=us
ibm-allgroups: cn=testers_and_programmers,o=ibm,c=us

dn: cn=jon,o=ibm,c=us
ibm-allgroups: cn=staticgroup,o=ibm,c=us
ibm-allgroups: cn=programmers,o=ibm,c=us
ibm-allgroups: cn=testers_and_programmers,o=ibm,c=us

dn: cn=diane,o=ibm,c=us
ibm-allgroups: cn=testers,o=ibm,c=us
ibm-allgroups: cn=testers_and_programmers,o=ibm,c=us

dn: cn=programmers,o=ibm,c=us
ibm-allmembers: cn=jon,o=ibm,c=us
ibm-allmembers: cn=saheem,o=ibm,c=us

dn: cn=testers,o=ibm,c=us
ibm-allmembers: cn=diane,o=ibm,c=us
ibm-allmembers: cn=jeff,o=ibm,c=us

dn: cn=staticgroup,o=ibm,c=us
ibm-allmembers: racfid=karen,profiletype=user,cn=racf
ibm-allmembers: cn=saheem,o=ibm,c=us
ibm-allmembers: cn=jon,o=ibm,c=us

dn: cn=testers_and_programmers,o=ibm,c=us
ibm-allmembers: cn=jon,o=ibm,c=us
ibm-allmembers: cn=saheem,o=ibm,c=us
ibm-allmembers: cn=diane,o=ibm,c=us
ibm-allmembers: cn=jeff,o=ibm,c=us
```

Figure 5-21 *ibm-allMembers and ibm-allGroups search results*

5.5.3 Static, dynamic, and nested group pros and cons

Although static groups are easy to define and setup, they can quickly become an administrative headache to manage when users need to be added or removed on a regular basis. This is when using a dynamic group might make sense. Instead of an LDAP administrator manually adding or removing users from a group, a dynamic group search expression could automatically find all users that ought to belong to the group. For example if `cn=jon,o=ibm,c=us` in Figure 5-20 on page 124 has changed jobs and should now be an tester instead of a programmer, the LDAP administrator only needs to change the title

attribute of `cn=jon,o=ibm,c=us` to `tester`. When this change is done, `cn=jon,o=ibm,c=us` is automatically added to the `cn=testers,o=ibm,c=us` dynamic group and is removed from the `cn=programmers,o=ibm,c=us` dynamic group.

Nested groups are useful when there is a need for a “super” group to contain everyone within a certain project. In Figure 5-20 on page 124, the `cn=testers_and_programmers,o=ibm,c=us` entry contains all the testers and programmers that are working on the same project. However, care needs to be taken when defining nested groups because they can become difficult to manage if deep hierarchies are introduced (e.g. Group A contains Group B which then contains Group C). A deep nested hierarchy can make it difficult for an LDAP administrator to correctly identify all members of a particular group.

5.5.4 Group gathering

After successfully authenticating to IBM Tivoli Directory Server for z/OS, the distinguished names of groups that the bound user belongs to are gathered and added to the bind information. If a GSSAPI (Kerberos) or SASL EXTERNAL bind resulted in any alternate DNs being obtained, the groups that these alternate DNs belong to are also gathered. These group DNs can then be used to set authorization rights within the LDAP directory. See 5.4, “Authorization using Tivoli Directory Server Access Control Lists (ACL)” on page 108 for more information.

Groups are gathered in the following manner:

- ▶ The back end or client operation plug-in extension that contains the bind DN is contacted to contribute DNs of any group entries that contain the bind DN or any of the alternate DNs. If authenticated to the SDBM back end, only groups in which the user ID's membership is active (has not been revoked) are included in the list. If the bind DN is not in a back end or a client operation plug-in extension (for example, after a Kerberos bind), this step is skipped.
- ▶ Each LDBM, CDBM, or TDBM back end that has **extendedGroupSearching on** specified in the LDAP server configuration file is also contacted to contribute the DNs of any group entries in the back end that contain the bind DN or any of the alternate DNs. The client operation plug-in extensions are also contacted to contribute group DNs if they have registered a `SLAPI_TYPE_GROUPS` callback type routine. Note that the SDBM back end does not support extended group searching.

Group gathering is not performed if any of the following is true:

1. The user has bound as any of the distinguished names specified in the **adminDN**, **peerServerDN**, and **masterServerDN** configuration options in the LDAP server configuration file.
2. The **authenticateOnly** server control (OID 1.3.18.0.2.10.2) is specified on the bind request. If using the SDBM back end for authentication only purposes, consider using the **authenticateOnly** server control to help streamline RACF authentication and bypass its group gathering. However this is not necessary if there is no LDBM, TDBM, GDBM, or CDBM back ends configured.

Notes:

If using IBM Tivoli Directory Server for z/OS V1R11 and earlier, the groups are determined during authentication.

If using IBM Tivoli Directory Server for z/OS V1R12 or later and LDAP password policy is active, the groups are determined during authentication time. If LDAP password policy is not active, the groups are determined at the beginning of the next non-bind request.

5.6 Password Policy

An important aspect of security when using passwords is the ability to define and administer a password policy. Password policies govern things like when and if passwords expire, how wrong passwords are handled, and rules for changing passwords. In LDAP, the administrator has the ability to define a global password policy for all users. Exceptions to the global policy can be made through the definition of group or individual password policies.

Note: The z/OS IBM Tivoli Directory Server password policy rules only apply to entries that have a **userPassword** value stored in a TDBM, LDBM, or CDBM back end. The server compatibility level must be 6 or greater and the CDBM back end must be configured to use LDAP password policy

5.6.1 Multiple password policies

In z/OS v1R12 LDAP, global password policy, group password policy and individual password policy support has been added with the CDBM back end. When determining which set of password rules a user should adhere, all three types of policies, if they exist, will be taken into consideration.

Global password policy

All users in the LDBM, TDBM, or CDBM back end not participating in native authentication are forced to comply with the rules defined in global password policy entry. When the global password policy entry is created by a server, the attribute **ibm-pwdPolicy** is set to FALSE, which means all password policy entries will be ignored by the server. Only when the attribute is set to TRUE will the password rules be enforced by the server.

An auxiliary object called **ibm-pwdGroupAndIndividualPolicies** is introduced. This object class can only be added to the global password policy entry. The only attribute in this object class would be the "MUST" attribute **ibm-pwdGroupAndIndividualEnabled**. If the global password policy is turned ON, a value of TRUE indicates that global, group and individual password policies are to be considered when evaluating password policy. A value of FALSE indicates that only the global password policy is used. The default value is FALSE.

Group password policy

With a group password policy, an association between a group object and a password policy entry is introduced so that the members of the group can be controlled by a set of special password rules. A new operational attribute, **ibm-pwdGroupPolicyDN** pointing to a password policy entry can be used in any user group objects, such as **accessGroup**, **accessRole**,

groupOfNames, groupOfUniqueNames, ibm-nestedGroup, ibm-dynamicGroup and ibm-staticGroup.

Because a user entry can belong to more than one group, multiple group password policy entries might need to be evaluated before the user's group policy can be determined.

Attributes in all the group password policy entries are combined to form a union of attributes with the most restrictive attribute values taking precedence. By associating a value of `cn=noPwdPolicy` with attribute **ibm-pwdGroupPolicyDN** for a password policy extended group entry, an administrator can exempt that group's policy from being used in the evaluation of the composite group policy. This means if a user belongs to a group to which a `cn=noPwdPolicy` value is assigned, the user's effective policy will not include any attributes from this group policy. Other group policies and the global policy and the individual policy will still be evaluated.

Note: If a user belongs to one or more group(s) that have `cn=noPwdPolicy` specified, belongs to no groups for which the **ibm-pwdGroupPolicyDN** attribute refers to a valid password policy entry, and has no individual password policy, then the user is exempt from any password policy rules.

Individual password policy

With an individual password policy, every user is allowed to have his or her own password policy. With a new operational attribute, **ibm-pwdIndividualPolicyDN** pointing to a password policy entry, a user entry will have its own password policy entry. This named reference password policy design provides an easy way to associate multiple user entries to the same policy entry. By changing the attributes of the password policy entry, an administrator can effectively manage a set of users without modifying any of the user entries.

By assigning a value of `cn=noPwdPolicy` to attribute **ibm-pwdIndividualPolicyDN** for a password policy extended user entry, an administrator can exempt a user from any password policy controls. This is different from not defining the attribute in the entry. If the attribute is not defined, the user's effective password policy will be derived from the user's group, if it exists, and the global policy. However, if the attribute is defined with the special value, then the effective password policy will not be evaluated at all and the user will not be controlled by any password rules.

Note: Not all of the password policy attributes need to be defined in a user's individual or group password policy entry. During password policy evaluation time, a user's individual, group and global password policy are searched in that order. If an attribute is not defined in the individual password policy entry, it will be searched in the composite group password policy entry. If it is not found, the attribute in the global password policy entry will be used. In case the attribute is not defined in the global password policy entry, then a default value will be assumed.

5.6.2 Meaning of various attributes in password policy

This section defines the various password policy attributes and operational attributes..

Password policy attributes

- ▶ **ibm-pwdPolicyStartTime:** This attribute contains the time when the password policy was turned ON.

- ▶ **pwdAttribute:** This attribute specifies the name of the attribute to which the password policy is being applied, and can only be set to the **userPassword** attribute.
- ▶ **pwdMinAge:** This attribute specifies the number of seconds that must pass since the last password modification before modifying a password.
- ▶ **pwdMaxAge:** This attribute specifies the number of seconds after which a modified password will expire (0 means password does not expire).
- ▶ **pwdInHistory:** This attribute specifies the number of passwords that are stored in the **pwdHistory** attribute.
- ▶ **pwdCheckSyntax:** This attribute indicates whether or not the password will be checked for syntax (0 means syntax checking will not be enforced. 1 means the server will check the syntax, and if the server is unable to check the syntax (due to a hashed password or other reasons) it will be accepted. 2 means the server will check the syntax, and if the server is unable to check the syntax it returns an error refusing the password).
- ▶ **pwdMinLength:** This attribute specifies the minimum length of the password string.
- ▶ **pwdExpireWarning:** This attribute specifies the maximum number of seconds before a password is about to expire that expiration warning messages will be returned to an authenticating user.
- ▶ **pwdGraceLoginLimit:** This attribute specifies the number of times an expired password can be used to authenticate a user.
- ▶ **pwdLockoutDuration:** This attribute specifies the number of seconds that the password cannot be used to authenticate due to specified **pwdMaxFailure** failed bind attempts.
- ▶ **pwdMaxFailure:** This specifies the maximum number of consecutive failed bind attempts allowed, after which the password cannot be used to authenticate (0 means the value of **pwdLockout** will be ignored).
- ▶ **pwdFailureCountInterval:** This attribute specifies the number of seconds after which the password failures are removed from the failure counter even though no successful authentication has occurred.
- ▶ **passwordMinAlphaChars:** This attribute specifies the minimum number of alphabetic characters that the password string must have. If the server is unable to check the number of alphabetic characters, then the server might continue processing, depending on the value of the **pwdCheckSyntax** attribute.
- ▶ **passwordMinOtherChars:** This attribute specifies the minimum number of numeric and special characters that the password string must have. If the server is unable to check the number of other characters, then the server might continue processing, depending on the value of the **pwdCheckSyntax** attribute.
- ▶ **passwordMaxRepeatedChars:** This attribute specifies the maximum number of times a given character can be used in a password. If the server is unable to check the actual password characters, then the server might continue processing, depending on the value of the **pwdCheckSyntax** attribute.
- ▶ **passwordMinDiffChars:** This attribute specifies the minimum number of characters in the new password that must be different from the characters in the old password. If the password has been one-way encrypted, the server will be unable to check actual password characters. The server might continue processing, depending on the value of the **pwdCheckSyntax** attribute.
- ▶ **ibm-pwdPolicy:** This attribute specifies whether the password policy is turned ON or OFF.
- ▶ **pwdLockout:** This attribute indicates whether or not a password can be used to authenticate after a specified number of consecutive failed bind attempts.

- ▶ **pwdAllowUserChange:** This attribute specifies whether or not the users are allowed to change their own passwords.
- ▶ **pwdMustChange:** This attribute specifies whether or not the users must change their password when they first bind to the directory after the administrator has reset their password.
- ▶ **pwdSafeModify:** This attribute specifies whether or not the existing password must be sent when changing a password.
- ▶ **ibm-pwdGroupAndIndividualEnabled:** This attribute determines if the Group Password policies and the Individual Password policies have to be considered or not during the Effective Password policy evaluation.

Password policy operational attributes

There are a number of password policy operational attributes that are stored in user entries subject to password policy that keep track of password policy state information. These attributes are:

- ▶ **pwdAccountLockedTime:** Contains the time at which the account was locked. If the account is not locked, this attribute is not present.
- ▶ **pwdChangedTime:** Contains the time the password was last changed or the password policy start time whichever is recent.
- ▶ **pwdExpirationWarned:** Contains the time at which the password expiration warning was first sent to the client.
- ▶ **pwdFailureTime:** A multi-valued attribute containing the times of previous consecutive login failures. If the last login was successful, this attribute is not present.
- ▶ **pwdHistory:** A multi-valued attribute containing a history of the previous password values used.
- ▶ **pwdGraceUseTime:** A multi-valued attribute containing the times of the previous grace logins.
- ▶ **pwdReset:** Contains the value TRUE if the password has been reset and must be changed by the user. The value is otherwise FALSE or not present.
- ▶ **ibm-pwdAccountLocked:** Indicates that the account has been administratively locked.
- ▶ **ibm-pwdIndividualPolicyDn:** DN of a password policy entry that can be associated with a user entry.
- ▶ **ibm-pwdGroupPolicyDn:** DN of a password policy entry that can be associated with a group entry.

Password policy checking

The z/OS IBM Tivoli Directory Server password policy is checked during;

- ▶ Pre-bind and pre-compare operations involving the **userPassword** attribute value to ensure that the password has not expired or the user's account has not been locked from authenticating to the directory.
- ▶ Post-bind and post-compare operations involving the **userPassword** attribute value to record failure login, clearing failure login time only if password is not expired and operation is successful.
- ▶ Pre-add and pre-modify operations involving the **userPassword** attribute value to ensure that the password is compliant with minimum and maximum number of characters that are given in the password, the password is allowed to be changed at this time, and if the password must be changed at this time for the user.

- ▶ Post-add and post-modify operations involving the `userPassword` attribute value to set the account reset flag after the administrator resets the password and `pwdMustChange` is true.

Native Authentication and expired passwords

In general, LDAP Password Policy does not apply to the SDBM back end or to native authentication. Password Policy for those users is determined by the underlying security system.

However, support has been added that allows a native authentication user to change their expired password on a modify delete-add operation. In older releases of LDAP, an expired password caused the ldap bind to fail. The only way to change the password was to check for the bind reason code of R004109 The password has expired and to bind again, this time with the `oldpasswd/newpasswd` syntax on the bind. This was contrary to the common practice of allowing the modify delete/add to change the password.

Now in this situation, if the server has been configured with the server config option `nativeUpdateAllowed reset`, and if the password policy control is sent on the LDAP bind, the bind will succeed with a response control called `changeAfterReset`. This then indicates that a modify request with delete of the old password and add of newpassword will succeed. If any other modify request is made, it will still fail with password expired.

These are the items that need to be set for this support to work:

- ▶ In the Server Configuration file, along with other native authentication options, set `nativeUpdateAllowed` to `reset`
- ▶ Send the Password Policy Control on the ldap bind
- ▶ Check for the `changeAfterReset` response in the control from bind
- ▶ Do an LDAP modify with the delete-add format

The z/OS `ldapmodify` client command is set up to provide this functionality. The command always sends the `PasswordPolicy` control and, if the z/OS IBM Tivoli Directory Server server is configured properly, the command will check the bind response and allow the modify delete-add operation change the password.

For example:

```
ldapmodify -D cn=user1,c=ca -w secret -f del_add.ldif
```

The contents of `del_add.ldif` are shown in Figure 5-22.

```
dn: cn=user1,c=ca
changetype: modify
delete: userpassword
userpassword: secret
-
add: userpassword
userpassword: abcdef
```

Figure 5-22 `del_add.ldif`

Sample C language code is provided in Appendix B, “Sample C code” on page 305, as an example of using the API to recognize an expired password on a successful `ldap_bind` of a native authentication user and to change the password with an `ldap_modify`.

5.7 Encryption and Hashing

IBM Tivoli Directory Server for z/OS supports a number of encryption and hashing algorithms to provide secure protection for sensitive passwords and other data that reside in LDBM, TDBM, and CDBM back end entries.

The **pwEncryption** option in the LDBM, TDBM, and CDBM back end sections of the LDAP server configuration file specifies the encryption or hashing method that is used to protect **userPassword** attribute values. The following values are supported on the **pwEncryption** option:

- ▶ AES:*keylabel* (Advanced Encryption Standard) where *keylabel* specifies a key in ICSF
- ▶ DES:*keylabel* (Data Encryption Standard) where *keylabel* specifies a key in ICSF
- ▶ SHA (SHA-1)
- ▶ SSHA (Salted SHA-1 – only supported in z/OS V1R12 and later)
- ▶ crypt
- ▶ md5
- ▶ none (default option)

The **secretEncryption** option in the LDBM, TDBM, and CDBM back end sections of the LDAP server configuration file specifies the encryption that is used to protect the **secretKey**, **replicaCredentials**, **ibm-replicaKeyPwd**, and **ibm-slappMasterPw** attribute values. AES, DES, and having no encryption is supported.

SHA, SSHA, crypt, and md5 are one-way hashing algorithms, whereas AES and DES are two-way encryption algorithms.

When a value is hashed in an one-way hashing algorithm, the resulting hash is always the same and the original value cannot be obtained unless you do a dictionary attack to determine the original un-hashed value. To help prevent these dictionary attacks, the Salted SHA-1 hashing algorithm was introduced in z/OS V1R12 to help randomize the resulting final hash. This gives the impression that the underlying clear value is not the same when in fact it is.

When using the crypt algorithm there are two considerations that need to be taken into account:

1. The crypt algorithm only accepts the first 8 characters of a password value when authenticating or storing in an LDBM, TDBM, or CDBM entry. If you need to use longer passwords, use another hashing or encryption method.
2. There are two crypt algorithms supported in IBM Tivoli Directory Server for z/OS: an ASCII-based and an EBCDIC-based algorithm. The ASCII-based crypt algorithm should be used when adding or bulkloading **userPassword** values from LDAP servers running on an ASCII platform so that authentications work properly on other LDAP servers. However, the EBCDIC-based crypt algorithm should be used if you previously migrated from the z/OS Integrated Security Services LDAP server and have not migrated all **userPassword** attribute values to use the ASCII-based crypt algorithm or another encryption or hashing method. The crypt algorithm that is used is controlled by the setting of the **pwCryptCompat** option in the LDBM, TDBM, or CDBM back end. When not specified in the back end or set to on, the EBCDIC-based crypt algorithm is used. Otherwise the ASCII-based crypt algorithm is used.

AES and DES are two-way encryption algorithms that can use keys defined in the ICSF CKDS (Cryptographic Key Data Set). When encrypting data with a two-way encryption algorithm, it is possible to decrypt the data to obtain the original un-encrypted value. The keys used for AES or DES encryption can be generated using the ICSF KGUP utility and stored in the

ICSF CKDS. Depending on the hardware and ICSF level available on your system, the AES and DES keys can be in the clear or encrypted. DES keys in the ICSF CKDS can be single-length (56 bit), double-length (112 bit), or triple-length (168 bit). AES keys can be 128 bytes, 192 bytes, or 256 bytes long in the ICSF CKDS.

Figure 5-23 shows examples that can be used for generating various AES or DES keys with the ICSF KGUP utility.

```
ADD TYPE(DATA) LENGTH(32) ALGORITHM(AES),  
    LABEL(AES256.REDBOOK.ENCRYPTED)  
  
ADD TYPE(CLRAES) LABEL(AES256.REDBOOK.CLEAR),  
    KEY(0123456789ABCDEF,0123456789ABCDEF,  
    0123456789ABCDEF,0123456789ABCDEF)  
  
ADD TYPE(DATA) ALGORITHM(DES) LENGTH(8),  
    LABEL(DES56.REDBOOK.ENCRYPTED)  
  
ADD TYPE(CLRDES) LABEL(DES56.REDBOOK.CLEAR),  
    KEY(0123456789ABCDEF)
```

Figure 5-23 Examples of creating AES and DES keys with the ICSF KGUP utility

After running the ICSF KGUP utility, make certain to refresh the CKDS so that the new AES and DES keys are available. To use the AES256.REDBOOK.CLEAR AES key on the **pwEncryption** or **secretEncryption** configuration option specify:

```
AES:AES256.REDBOOK.CLEAR
```

```
RDEFINE CSFKEYS AES256.REDBOOK.CLEAR UACC(NONE)  
PERMIT AES256.REDBOOK.CLEAR CLASS(CSFKEYS) ACCESS(READ) ID(GLDSRV)  
SETROPTS RACLIST(CSFKEYS) REFRESH
```

Figure 5-24 Refreshing ICSF CKDS

See *z/OS V1R12.0 ICSF Overview*, SA22-7519 for more information about the AES and DES keys that are supported for your hardware and ICSF level. See *z/OS V1R12 ICSF Administrator's Guide*, SA22-7512 for more information about using the KGUP utility to store keys for AES and DES encryption.

When the **userPassword**, **secretKey**, **replicaCredentials**, **ibm-replicaKeyPwd**, and **ibm-slapdMasterPw** attribute values are stored in the LDBM, TDBM, or CDBM back ends, the values are tagged with the encryption or hashing algorithm used (for example SSHA or AES:AES256.REDBOOK.CLEAR). When the **pwEncryption** or **secretEncryption** configuration options are changed to a new encryption or hashing algorithm, the values hashed or encrypted in the old method are still usable. See 5.2, "Authentication mechanisms supported by IBM Tivoli Directory Server for z/OS" on page 92 for information about how authentications occur with different hashing or encryption methods.

5.8 SSL/TLS

IBM Tivoli Directory Server for z/OS can be configured to use SSL (Secure Sockets Layer) and TLS (Transport Layer Security) protocols to provide protected communications between the LDAP server and client applications. This is especially important when sensitive

information, such as the user's bind distinguished name and password, is exchanged between the LDAP server and client applications. An SSL/TLS connection between the server and client can be used to encrypt the connection to make it more difficult to obtain this sensitive data and help prevent man-in-the-middle attacks. Authorization to entries in the LDBM, TDBM, and CDBM back ends can also be protected based on whether an encrypted connection is used during an SSL/TLS connection. See 5.4, "Authorization using Tivoli Directory Server Access Control Lists (ACL)" on page 108 for more information about setting up access control filters.

The SSL and TLS protocols use PKI (public-key infrastructure) to establish and maintain the encrypted communications between the LDAP server and client applications. The z/OS Cryptographic Services SSL component of z/OS provides the services necessary for communicating with SSL/TLS.

When establishing an SSL/TLS connection between IBM Tivoli Directory Server for z/OS and the LDAP client application, the LDAP server transmits a certificate to verify its identity to the LDAP client. The client can optionally transmit its own certificate to the LDAP server to verify its identity. Both the LDAP server and client must verify that the certificates that they receive are valid by comparing the digital signature in the certificates with a signature that it computes based on having the public-key signer of the certificate. The LDAP server and client do this by reading a file or repository that contains these public keys or certificates.

These public keys or certificates are associated with the signers of the certificates and in z/OS they can be stored in an SSL key database file, RACF (SAF) key ring, or a PKCS#11 token. Generally, verifying a certificate requires another certificate that is known as the signing certificate or public key. Therefore, it is possible to introduce a chain of certificates where each certificate needs to be verified by its signing certificate until all certificates are verified. A certificate can be defined as a root certificate, which is a self-signed certificate. A self-signed certificate contains the public-key that was used to sign the certificate.

The key repository used by either the LDAP server or client must contain enough certificates to verify the certificates exchanged during the initial SSL/TLS handshakes to establish the encrypted connection. If any of these certificates are self-signed, they must be stored in the other's key repository. Similarly, if the certificates are signed by another certificate signer, then the signer's certificate and any certificates that this certificate depends upon must be stored in the key repository.

See *Implementing PKI Services on z/OS*, SG24-6968 for more information about how certificates and PKI work.

5.8.1 Certificates and key repositories

As mentioned earlier, there are three certificate key repositories supported by z/OS and IBM Tivoli Directory Server for z/OS:

- ▶ RACF(SAF) key ring: Key rings and certificates are created, stored, and managed in RACF by using the RACDCERT command.
- ▶ SSL key database file: Key database files and certificates are created, stored, and managed by a file in the UNIX System Services file system with the gskkyman utility.
- ▶ PKCS#11 tokens (ICSF): PKCS#11 tokens are stored in ICSF (Integrated Cryptographic Security Facility) but are managed using the RACF RACDCERT command or the gskkyman utility.

The RACDCERT command provides the ability to let the RACF database store the key repository and certificates for SSL. The key repository in RACF is called the key ring and

contains the certificates. The RACDCERT command provides the ability to create certificates, import and export certificates (public keys) to and from other repositories, and change existing certificate information. The following are the main RACDCERT subcommands that are used when creating a RACF key ring repository:

- ▶ RACDCERT ADD: Imports in a certificate and assigns it to a RACF user.
- ▶ RACDCERT ADDRING: Creates a RACF key ring
- ▶ RACDCERT CONNECT: Connects a certificate to a RACF key ring.
- ▶ RACDCERT GENCERT: Used to generate a certificate
- ▶ RACDCERT LIST: Lists the contents of a certificate.
- ▶ RACDCERT LISTRING: Lists the certificates connected to a RACF key ring.

See 5.2.6, “External (SSL)” on page 101 for an example of creating a RACF key ring and a certificate for the LDAP server. For more information about these RACDCERT subcommands and the remaining RACDCERT subcommands, see *z/OS V1R12.0 Security Server RACF Command Language Reference, SA22-7687*.

The **gskkyman** utility is a z/OS OMVS shell based menu-driven utility shipped by System SSL. This utility manages a password-protected file called the key database file in the UNIX System Services file system. The key database file contains the private and public keys used by certificates connected to the file. The **gskkyman** utility provides the ability to create certificates, import and export certificates (public keys) to and from other repositories, and change existing certificate information. See Chapter 10 of *z/OS V1R12.0 System SSL Programming, SC24-5901* for more information about the **gskkyman** utility.

As mentioned earlier, either the RACDCERT command or the **gskkyman** utility can be used to create or manage PKCS#11 tokens that are stored in ICSF. These PKCS#11 tokens can contain certificates that are used by either IBM Tivoli Directory Server for z/OS or the LDAP client. The issuer of the RACDCERT command or **gskkyman** utility must have access to the CRYPTO SAF class to perform operations against the PKCS#11 token. See the above references for more information about managing PKCS#11 tokens in ICSF.

5.8.2 Setting up IBM Tivoli Directory Server for z/OS to use SSL/TLS

IBM Tivoli Directory Server for z/OS has a number of SSL configuration options that must be specified to setup the LDAP server to use SSL/TLS protected communications. These configuration options are specified in the global section of the LDAP server's configuration file.

- ▶ **listen**
- ▶ **sslAuth**
- ▶ **sslCertificate**
- ▶ **sslCipherSpecs**
- ▶ **sslKeyRingFile**
- ▶ **sslKeyRingFilePW**
- ▶ **sslKeyRingPWStashFile**
- ▶ **sslMapCertificate**

The multi-valued **listen** configuration option indicates the port and optionally the host name or IP address in LDAP URL format (`ldap[s]://[hostname | IP address]:[port]`) that the LDAP server should listen for incoming client requests on.

- ▶ When `ldap://` is specified as the URL prefix, it indicates that the LDAP server will listen for either non-secure or secure communications on the port and TCP/IP interface specified. By default, the LDAP server accepts only non-secure communications on these ports, but it can be changed to a secure connection using the StartTLS extended operation. Following the StartTLS extended operation, a secure SSL/TLS connection is negotiated

and the connection is secured until the connection is broken, the user unbinds or disconnects from the LDAP server, or a StopTLS extended operation is done. The StopTLS extended operation results in a non-secure connection being re-established between the client and server.

- ▶ When `ldaps://` is specified as the URL prefix, it indicates that the LDAP server will only listen for secure communications on the port and TCP/IP interface specified.

The `sslKeyRingFile` configuration option specifies the name of the RACF key ring, SSL key database file, or PKCS#11 token that is used by the LDAP server. If using a PKCS#11 token, it must be specified as `*TOKEN*/NAME` where `NAME` is the name of the PKCS#11 token.

If the `sslKeyRingFile` configuration option specifies a SSL key database file, either the `sslKeyRingFilePW` or `sslKeyRingPWStashFile` configuration option must be specified. The `sslKeyRingFilePW` option specifies the password assigned to the SSL key database file, and the `sslKeyRingPWStashFile` specifies a stash file that better protects the password for the key database file. The stash file is created using the `gskkyman` utility.

The `sslCertificate` configuration option specifies the label of the certificate to use in the SSL key database, RACF key ring, or PKCS#11 token. If this option is not specified, the default certificate (if there is one) is used in the key database file, RACF key ring, or PKCS#11 token.

The `sslAuth` configuration option indicates whether server authentication or server and client authentication occurs.

- ▶ When set to `serverAuth`, the LDAP server sends the client its certificate, which the client then validates. If the client successfully validates the server's certificate during the handshake, secure communications begin between the server and client.
- ▶ When set to `serverClientAuth`, both the LDAP server and client exchange certificates and they must both be verified. After both certificates are verified, secure communications begin between the server and client.

To perform SASL EXTERNAL binds, the `sslAuth` option must be set to `serverClientAuth`. For more information about SASL EXTERNAL binds, see 5.2.6, “External (SSL)” on page 101.

The `sslCipherSpecs` configuration option specifies the ciphers that are accepted from connected clients. If this option is not specified, then all ciphers accepted by the LDAP server are used.

The `sslMapCertificate` configuration option indicates if the client certificate ought to be mapped to a RACF user when doing a SASL EXTERNAL bind. For information about how SASL EXTERNAL bind mapping works, see 5.2.6, “External (SSL)” on page 101.

5.9 Persistent Search

Often application are designed to fetch data from back end systems and store them in application-specific cache memory to speed up response times and enhance the user experience. While doing so, they also need to take care of refreshing the data in cache and making sure that stale data is not presented to the user. Maintaining stale data in the cache can potentially impact business critical applications.

As its name implies, the Persistent Search feature of IBM Tivoli Directory Server is a search operation that continues after the initial set of matching entries is returned. This feature allows

you to perform a search, gather the results of the search operation, and then, whenever an entry in the result set is modified, get a new copy of that entry.

Persistent Search is an extension to the LDAP v3 search operation that moves the burden of checking for updates within a search result set from the client to the server. The Persistent Search control allows the client to perform a normal LDAP search operation (specifying the base DN, scope of search, search filter, and so on) and then, rather than having the server return a SearchResultDone message at the end, the operation maintains a connection so the client can be updated each time an entry in the result set changes. This allows the client to maintain a cache of the entries it is interested in, or trigger logic whenever an update occurs.

In LDAP, a control is an extension mechanism that allows you to change the way an existing LDAP operation works. In this case, the control applies to the search operation. In addition to the normal things you specify when performing a search (like baseDN, scope, filter, and so on), this control lets you specify data that will help the server know how to process the Persistent Search.

The advantages of Persistent Search are:

► **Cache Consistency**

In an LDAP client application with high performance needs, you might want to maintain a temporary, local cache of information obtained through an LDAP search operation. To improve performance, the local cache is always consulted before sending a request to an LDAP server. A Persistent Search request where the `changesOnly` flag is `False` can be used if it is desired to prime the cache. Otherwise, `changesOnly` would be set to `True` in the request.

Caches are also used for reasons other than performance improvement. In certain cases, they arise naturally out of a particular application's design. For example, an LDAP client designed for administration of information held in LDAP servers will undoubtedly generate window displays that show information gleaned from an LDAP server. The window display is a cache that is active and visible until the user of the application takes an action that causes other information to be displayed. A refresh button or similar control may be provided to users to allow them to update the cached display. A Persistent Search request can be used instead by the administrative application to automatically refresh the window display as soon as the underlying LDAP information changes.

► **Triggered Actions**

An LDAP client application might want to take action when an entry in the directory is changed. A Persistent Search request can be used to proactively monitor one or more LDAP servers for interesting changes that in turn cause specific actions to be taken by an application. For example, an electronic mail repository might want to perform a “create mailbox” task when a new person entry is added to an LDAP directory, and a “delete mailbox” task when a person entry is deleted from an LDAP directory.

Note: The Persistent Search operation is memory and connection-intensive for the LDAP server, as it not only needs to maintain an open TCP connection but also needs to maintain the search request associated with that connection. Every change that happens in the directory database needs to be evaluated against the search criteria of the Persistent Search client, and the client is notified of the change or not accordingly.



Reliability, availability, and scalability

This chapter discusses IBM Tivoli Directory Server reliability, availability, and scalability.

6.1 Reliability, Availability and Scalability

For most organizations, the reliability, availability, and scalability (RAS) of their systems are crucial characteristics that, if lacking, could be detrimental to the overall sustainability of the organization. IBM Tivoli Directory Server for z/OS has many features that specifically address many common RAS requirements. In previous sections, the overall IBM Tivoli Directory Server architecture and its use of DB2 highlighted the server's scalability. In the upcoming sections, we will discuss IBM Tivoli Directory Server's features that work towards ensuring the server's availability while allowing you to limit the amount of overall system resources that are consumed by the IBM Tivoli Directory Server server.

6.1.1 Availability

Many times a system's availability is enhanced using redundancy. For systems that manage data, redundancy can come in two forms: data redundancy and application redundancy. Data redundancy involves having duplicated data. Often business processes or perhaps the applications managing data ensure the duplicated data remains consistent across all copies. Application redundancy involves applications that have rigid availability requirements being duplicated across the enterprise. In both cases, the idea is that if a system running a mission critical application or storing critical data becomes unavailable, a transfer of operation can occur seamlessly to another system that houses the duplicated data, application, or both. If all goes well, the users of the data or applications should not even notice the shift. Similarly, if a system is overworked, its workload could be distributed to other systems housing duplicated data, applications, or both. Once again the availability of the system will be retained. IBM Tivoli Directory Server addresses availability in the same fashion. That is, it supports application redundancy using Sysplex and shared data, and data redundancy through its built-in replication model. These two features, in combination with something like VIPA, allow seamless transfer of LDAP workloads throughout the enterprise to ensure availability.

6.2 Sysplex

As shown in 1.4.2, “Multi-Server (Sysplex)” on page 8, IBM Tivoli Directory Server leverages z/OS's sysplex features to help ensure application availability. It relies on z/OS's XCF facility to allow multiple LDAP instances across many LPARS to communicate and work in concert as though they are all just one LDAP server. In this configuration, there is one sysplex master LDAP server instance and multiple sysplex replica LDAP server instances. When working with CDBM, LDBM or file-system-backed GDBM back ends, the directory data is initially shared using a shared UNIX Systems Services file system. At start time, the sysplex group master (the first server initialized) reads into memory the persistent data from the shared file system. When the sysplex group replicas initialize, they request a copy of the data read in by the sysplex group master. After initialization is complete, all updates to the data are coordinated through the sysplex group master. As a result, the managing of the persistent storage is only done by the sysplex group master. When working with TDBM back ends or DB2-backed GDBM back ends, the directory data is shared through DB2 data sharing. If advanced replication is enabled, similar to the file-system-backed back ends, all LDAP update requests are coordinated through the sysplex group master. If advanced replication is not enabled, each sysplex group member that handles an LDAP update request updates DB2 directly. See configuration examples in Chapter 10, “Using IBM Tivoli Directory Server in a Parallel Sysplex” on page 217 for details about configuring a sysplex.

If a virtual IP address configuration is setup, LDAP clients connect to a virtualized IP address. TCP/IP then handles connecting to one of the servers in the sysplex. It could be the sysplex

master or a sysplex replica. Depending on the operation, the server contacted might satisfy the client request using the shared data. Sometimes, if the server contacted is a sysplex replica, it might not be able to satisfy the client request due to IBM Tivoli Directory Server internal implementation requirements. The replica server will then use XCF messaging to communicate with the sysplex master instance and have the master instance satisfy the request. When it is done, the sysplex replica relays the results to the client. This is all transparent to the client. To ensure availability, if the master server instance becomes unavailable, another replica server instance is promoted to replica master.

Note: IBM Tivoli Directory Server can be configured to register with z/OS's Automatic Restart Manager (ARM) to automatically restart when an instance fails.

Because it is using shared data, IBM Tivoli Directory Server's sysplex support provides application redundancy only. In the upcoming section, we will show how IBM Tivoli Directory Server for z/OS relies on its replication feature to facilitate data redundancy.

6.3 Replication

Generally, LDAP servers provide a mechanism for replicating entries across multiple server instances. The basic idea is the directory is duplicated across multiple LDAP servers participating in a replication configuration. When a client updates the directory in a server, either by adding, deleting, moving, or modifying LDAP entries, the update is replicated using the LDAP protocol to all participating servers. Note that only certain servers can accept update requests from clients. Other servers will only service search requests and refer all client update requests to other servers that can accept updates.

IBM Tivoli Directory Server for z/OS provides two options for replication. One model is referred to as basic replication. This is the model available since IBM Tivoli Directory Server for z/OS became generally available. For IBM Tivoli Directory Server for z/OS R11, an advanced replication model was introduced. Among other things, the advanced replication feature allows a more granular replication model. That is, instead of maintaining an entire back end as is done with basic replication, advanced replication allows replication to be configured at a sub-tree level. Moreover, different sub-trees within a directory can take on different roles in the replication configuration. Because advanced replication has many more features, it will be the focus of this section.

Note that the process of replicating should not impact the client's interaction with the server. IBM Tivoli Directory Server for z/OS performs the replication activity on distinct threads of execution. To ensure this, IBM Tivoli Directory Server internally queues updates to replicate. Then another thread of execution retrieves updates from the queue and replicates it to the consumer. The queue of updates is maintained persistently. Thus, if IBM Tivoli Directory Server shuts down, when it is restarted it will continue to replicate the updates that had not been replicated prior to its shutdown.

Before going into details, we will highlight key terminology:

Replication Context	A sub-tree within the directory that participates in replication
Supplier	A general term for a server that will replicate any updates it receives for a replication context
Consumer	A general term for a server that will accept replicated updates from a supplier for a replication context

Referral	The LDAP return code and server addresses returned by the consumer server when a client attempts to update a context
Replication Agreement	An LDAP entry that defines the details required to configure a connection between a Supplier and a Consumer for a given replication context.
Master	A supplier that will accept updates from a client for a context. It will replicate updates from clients to all consumers it has an agreement defined with.
Peer	A supplier and a consumer that will accept updates from a Client and be a consumer of other peer or master updates for a context. It will replicate updates from clients to all consumers it has an agreement defined with. It will NOT replicate updates from another peer or master.
Read Only Replica	A consumer that contains replicated contexts, but will only accept updates from a supplier. Client requests will result in a referral to a Master or (less likely) a Peer.
Forwarder	A consumer that contains replicated contexts, but will only accept updates from a supplier. Client requests will result in a referral to a Master or (less likely) a Peer. Unlike the read only replica, this consumer replicates supplier updates for a context with agreements.
Gateway	A special peer that serves as an entry point for replicating updates within a distinct sub-network of servers. When connected to other peer gateways, it will replicate updates from the peer gateway to all of its consumers in the sub-network, but not to other gateway peers. Any updates it receives from its sub-network of servers will only be replicated to other peer gateways only. Any client updates will be replicated to all consumers, including peer gateways.
Topology	The type of configuration used when deploying a replication environment consisting of LDAP consumers and suppliers. There are four topologies: master - replica, peer - peer, cascading/forwarding, and gateway.

6.4 Topology

Prior to configuring IBM Tivoli Directory Server for advanced replication, the LDAP administrator should take the time to decide on what replication topology to deploy. You should consider the following factors:

- ▶ The complexity of the network or network(s) topologies where the various LDAP servers will be deployed
- ▶ The number of servers that should service client update requests
- ▶ The number of servers that should service search requests
- ▶ The “cost” of replicating updates across the network or networks
- ▶ The acceptable time that servers can remain out of sync

IBM Tivoli Directory Server for z/OS provides four replication topology options. The following sections summarizes the four topologies while giving guidance as to when the topology should be chosen.

Because the LDAP protocol is used between supplier and consumer, IBM Tivoli Directory Server for z/OS also supports replicating to LDAP servers on non-z/OS platforms. Certain features of IBM Tivoli Directory Server for z/OS might not be replicated due to compatibility issues. However, the common replication topologies can be set up such that updates from a master can be replicated to other LDAP servers on other platforms. In the upcoming section we will describe how to configure advanced replication on IBM Tivoli Directory Server for z/OS.

6.4.1 Master - Replica

This is the most common basic topology. The master accepts client requests and replicates updates to the read only replica. The read only replica will only accept updates from the master. This topology can be deployed for data redundancy purposes and to help reduce the search workload of the master server because clients can send search requests to both the replica and the master. However, because all updates are done over TCP/IP, there can be times when the replica and master are not synchronized. If the time the two servers are out of sync is a critical metric, analyze the network performance to determine if this topology is acceptable.

6.4.2 Peer - Peer

This is a common topology with more flexibility than the master - replica topology. Peers accept updates from clients and replicate the updates to each other. If a peer also has replication agreements to other consumers, it will replicate client updates to the others servers, but will not replicate updates to other peers. This topology can be deployed to provide a backup master server that can take over immediately if the primary master fails, and can help reduce the workloads of the servers because clients can send requests to all peers. However, because all updates are done over TCP/IP, there can be times when the peers are not synchronized. If the time the two servers are out of sync is a critical metric, analyze the network performance to determine if this topology is acceptable.

The peer-peer topology can also lead to conflicts. Consider if the same LDAP entry is being updated on two peers at the same time by clients. This will result in a conflict occurring when each server attempts to replicate to each other. IBM Tivoli Directory Server for z/OS provides an option to enable conflict resolution.

6.4.3 Forwarding/Cascading

This is an extension of the master - replica topology. In between a master and a replica is a forwarding/cascading server. The master accepts client requests and replicates updates to the cascading server. The cascading server will only accept updates from the master and then replicates those updates to the read only replica. The read only replica will only accept updates from the cascading server. This topology can be deployed to relieve replication workload from master servers in a network containing many widely distributed replicas. This topology can also be deployed to help reduce the search workload of the master server because clients can send search requests to all the servers. If the time the master and replica servers are out of sync is a critical metric, deploy this topology to reduce the latency.

6.4.4 Gateway

This topology is a complex topology intended for wide area networks where subnetworks can have replication topologies with many servers. The idea is to reduce network traffic among all the servers across the WAN. Instead the subnetworks communicate through Gateway peer

servers. Gateway peer servers will replicate updates from other Gateway peer servers to all servers in their sub-network only. On the other hand, a Gateway will replicate to other gateway peers any updates received from clients or servers within its sub-network. If necessary, in each sub-network, any synchronization latency can be reduced with one or more of the other topologies.

Note that for all of the listed topologies, when a server does not accept an update from a client, it is often configured to return a referral to the client. This referral can then be used by the client to redirect the request to a server that accepts client requests. Also note these topologies are not mutually exclusive. Clearly the Gateway topology would not be necessary unless it is deployed in conjunction with the other topologies.

6.4.5 Sysplex and Replication

As previously stated, sysplex is a mechanism for providing application redundancy, whereas replication is a mechanism for providing data redundancy. Note these two options are not mutually exclusive. That is, IBM Tivoli Directory Server server instances that are participating in a replication topology can also be set up as a sysplex group. IBM Tivoli Directory Server's replication support is sysplex-aware. Replication related requests against servers in a sysplex group will be properly coordinated among all of the servers in the group.

For suppliers that are set up as a sysplex group, the sysplex group master instance is responsible for sending replicated updates to consumers. If the sysplex group master shuts down, the IBM Tivoli Directory Server sysplex support automatically elects a new sysplex group member as the sysplex master. This server instance then will be responsible for doing all of the replication.

In a sysplex environment, all data is shared. This includes the replication queues. As a result, the new sysplex group master should seamlessly pick up where the former sysplex group master had left off prior to shutting down.

6.5 Setting up Replication

Prior to setting up advanced replication, a few updates are required in the IBM Tivoli Directory Server configuration file for all the server instances participating in the topology:

- ▶ In the global section of the file, the `serverCompatLevel` value must be 5 or higher.
- ▶ The suffixes of all the replication contexts must be defined as a suffix in the configuration file.

Note: There is no requirement for back end types of consumers and suppliers to be equivalent or for all contexts to exist in the same back end. For example, on a supplier the context can exist in a TDBM back end and be replicated to a consumer's LDBM back end.

- ▶ A CDBM back end must be defined and the `useAdvancedReplication` value should be on:

```
#CDBM
database cdbm GLDBCD31/GLDBCD64 cdbm
useAdvancedReplication on
```

After the configuration files have been updated and the LDAP server instances restarted, all of the remaining configuration is done by adding and modifying various LDAP entries in the

directory. The configuration can be separated into two sets: supplier configuration and consumer configuration. Recall suppliers can be master, peer, gateway, and cascading servers. Consumer servers can be peer, gateway, read only replica, and cascading servers. In the case of peers, gateways, and cascading servers, both supplier and consumer setup must occur. After updating the configuration file, a few more preliminary steps are required before moving ahead to consumer and supplier setup.

ServerID

For all servers participating in advanced replication, a server ID is required. The server ID is stored/set in the `cn=configuration` entry's `ibm-slapdServerId` attribute. It is a string attribute. By default, IBM Tivoli Directory Server for z/OS will generate a random entry-UUID value for this value. This value is needed for the rest of configuration and entry-UUIDs are not simple strings. The `cn=configuration` entry should be modified to set a `ibm-slapdServerId` that is easier to use when performing configuration.

SSL

All data is replicated unencrypted unless one way hashes are used for attributes like `userpassword`. Set up all consumer servers to support SSL connections so all suppliers can then use SSL connections to the consumers to help ensure data confidentiality.

Other replication-related configuration options

Advanced replication provides many features to manage replication topologies. The `cn=Replication`, `cn=Configuration`, `cn=Replication`, `cn=Log Management`, and `cn=Configuration` entries contain attribute settings that manage things like conflict resolution, error handling, and how queue status is displayed. Refer to *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for tuning the advanced replication feature.

Maintenance Mode

IBM Tivoli Directory Server for z/OS supports maintenance mode. In this mode, no non-admin client updates are accepted by a server. To ensure no client updates are lost while configuring replication, switch all servers to maintenance mode. This is done by issuing the following z/OS console command:

```
F procName,MAINTMODE ON
```

After configuration is complete, the following z/OS console command can be issued to activate all servers:

```
F procName,MAINTMODE OFF
```

After all preliminary steps are completed, the administrator must decide how the directory data in the suppliers and consumers will be initially synchronized. There are many questions to consider when deciding:

- ▶ Do any of the participating servers already contain the directory data?
- ▶ Are you starting with an empty directory that you will populate over time?
- ▶ If working with existing directory data that you would like to now replicate, how many entries or sub-trees will need to be remain synchronized?

As implied by the questions to consider, the configuration process has flexibility and thus can be varied. For sake of simplicity, the following sections will assume the directory data already exists in one supplier only and all servers are IBM Tivoli Directory Server for z/OS servers. LDIF files and the `ldapadd` or `ldif2ds` command can be used to populate the supplier, if it has not been done already. Also, there can be different commands and steps available than what

is described below. Refer to the IBM Tivoli Directory Server for z/OS publications for additional configuration options. Finally, configuring IBM Tivoli Directory Server for the distributed platforms is similar to the steps outlined here. However, differences do exist, especially in the areas of setting up SSL and setting the server ID. Refer to the IBM Tivoli Directory Server for distributed platforms publications for more information.

6.5.1 Consumer Configuration

The first step in setting up the consumer is establishing the supplier's bind DN and password. To perform this step it is necessary to know each replication context's root entry DN. Figure 6-1 shows how to define two replication contexts' supplier bind credentials.

```
dn: cn=Supplier Master1, cn=configuration
cn: Supplier Master
ibm-slapdMasterDN: cn=master1
ibm-slapdMasterPW: secret
ibm-slapdReplicaSubtree: o=Olympia,o=sampleLDBM
objectclass: ibm-slapdSupplier

dn: cn=Supplier Master2, cn=configuration
cn: Supplier Master
ibm-slapdMasterDN: cn=master2
ibm-slapdMasterPW: secret
ibm-slapdReplicaSubtree: o=Olympia,o=sampleTDBM
objectclass: ibm-slapdSupplier
```

Figure 6-1 Define two replication contexts' supplier bind credentials

After the supplier bind credentials are established, the consumer's directory might need to be primed. That is, if a replication context is not a defined suffix in the configuration file, all the parent entries of the context, up to the suffix must exist in the consumer's directory. If, in the example used here, the contexts are not suffixes, suffix entries `o=sampleLDBM` and `o=sampleTDBM` must exist in the consumer's directory.

Finally a referral list should be established in case client's direct updates to a read only replica and cascading servers. In our example, we will set the referral list on the supplier.

6.5.2 Supplier Configuration

Supplier configuration is not as straightforward as consumer configuration. The first step is to set the replication contexts. As stated, we are assuming the directory data already exists in one supplier. Thus we must modify the root entry of the replication context to add the **ibm-replicationContext** object class as shown in Figure 6-2.

```

dn:o=Olympia,o=sampleTDBM
changetype: modify
add: objectclass
objectclass: ibm-replicationContext

dn:o=Olympia,o=sampleLDBM
changetype: modify
add: objectclass
objectclass: ibm-replicationContext

```

Figure 6-2 Add the `ibm-replicationContext` objectclass

After the replication contexts have been set, various other LDAP entries must be added. First, add an **ibm-replicaGroup** entry as shown in Figure 6-3.

```

dn: ibm-replicaGroup=default, o=Olympia,o=sampleLDBM
objectclass: top
objectclass: ibm-replicaGroup
ibm-replicagroup: default

dn: ibm-replicaGroup=default, o=Olympia,o=sampleTDBM
objectclass: top
objectclass: ibm-replicaGroup
ibm-replicagroup: default

```

Figure 6-3 `ibm-replicaGroup` entry

The **ibm-replicaGroup** must be directly beneath the **ibm-replicationContext** object. It only serves as a grouping entry. Conceivably, one could group supplier to consumer relationships logically and have them represented in the directory using multiple **ibm-replicaGroup** objects.

The next entry needed is an **ibm-replicaSubentry** object. Unlike **ibm-replicaGroup** entries, **ibm-replicaSubentry** objects are critical for determining a server's role in replication.

```

dn: cn=master,ibm-replicaGroup=default, o=Olympia,o=sampleLDBM
objectclass: top
objectclass: ibm-replicaSubentry
ibm-replicationserverismaster: true
description: master server
ibm-replicaserverid: thisServerID

dn: cn=master,ibm-replicaGroup=default, o=Olympia,o=sampleTDBM
objectclass: top
objectclass: ibm-replicaSubentry
ibm-replicationserverismaster: true
description: master server
ibm-replicaserverid: thisServerID

```

Figure 6-4 `ibm-replicaSubentry`

The two critical attributes in an **ibm-replicaSubentry** object are **ibm-replicationserverismaster** and **ibm-replicaserverid**. For a server to serve as a supplier, the **ibm-replicaserverid** value must be equal to the **ibm-slapdServerId** value in the

cn=configuration entry. If these values do not match, the server will not behave as a supplier. If they do match, the **ibm-replicationserverismaster** is the attribute that distinguishes a cascading server from a master/peer server. If the value is FALSE, the supplier is a cascading server. Otherwise its a master or peer. This dictates how the server will react to client updates.

Immediately below the **ibm-replicaSubentry** object, one **ibm-replicationAgreement** object is needed for every consumer. See Figure 6-5.

```
dn: cn=aggr1,cn=master,ibm-replicaGroup=default, o=Olympia,o=sampleLDBM
objectclass: top
objectclass: ibm-replicationAgreement
ibm-replicaconsumerid: consumerServerID
ibm-replicaurl: ldaps://consumer1.abc.com:669
ibm-replicacredentialsdn: cn=simple1,cn=localhost
description: consumer connection

dn: cn=aggr1,cn=master,ibm-replicaGroup=default, o=Olympia,o=sampleTDBM
objectclass: top
objectclass: ibm-replicationAgreement
ibm-replicaconsumerid: consumerServerID
ibm-replicaurl: ldaps://consumer2.abc.com:669
ibm-replicacredentialsdn: cn=simple2,cn=localhost
description: consumer connection
```

Figure 6-5 **ibm-replicationAgreement**

The key attributes in these entries are **ibm-replicaurl** and **ibm-replicacredentialsdn**. The **ibm-replicaurl** defines the LDAP URL of the consumer. In the above examples, we are using **ldaps** to stress the importance of using SSL when replicating sensitive data. The **ibm-replicacredentialsdn** attribute defines the DN of the entry that contains the credentials for binding to the consumer. The reason the credentials are not included in the agreement entry is that the agreement entry, with all of the other entries shown in this section, ultimately is replicated to consumers. Because the bind credentials for a consumer allow full access to the consumer, it might be prudent to limit the exposure, especially when a supplier must connect to many consumers. Further, the possibility of replicating in the clear without SSL could lead to the credentials being compromised. In our example, we store the credential objects under **cn=localhost**. **cn=localhost** can never set up as a replication context.

The DN must point to an object with one of the following objectclasses: **ibm-replicationCredentialsExternal** or **ibm-replicationCredentialsSimple**. The **ibm-replicationCredentialsExternal** credentials object requires an SSL connection to the consumer and will force the supplier to bind to the consumer using the SASL EXTERNAL mechanism. The **ibm-replicationCredentialsSimple** entry will contain information for a simple bind to the consumer. See Figure 6-6.

```

dn: cn=simple1,cn=localhost
objectclass: ibm-replicationCredentialsSimple
cn: zForwarder simpl
replicaBindDN: cn=master1
replicaCredentials: secret

dn: cn=simple2,cn=localhost
objectclass: ibm-replicationCredentialsSimple
cn: zForwarder simpl
replicaBindDN: cn=master2
replicaCredentials: secret

```

Figure 6-6 ibm-replicationCredentialsSimple

Note the **replicaBindDN** and **replicaCredentials** attributes listed match the **ibm-slapdMasterDN** and **ibm-slapdMasterPW** attributes shown in the Consumer section above.

If more than one supplier will participate in the topology, **ibm-replicagroup**, **ibm-replicaSubentry** and **ibm-replicaAgreement** entries required by other suppliers can be added to the initial supplier being configured (assumed in this example to contain all data to replicate). Care must be taken to set the correct **ibm-replicaserverid** and **ibm-replicacredentialsdn** attribute values because for every supplier they could be separate, especially in the case of the server ID. Further, the credentials entries will be needed on each supplier.

6.5.3 Synchronizing the servers

After the servers have been properly configured and the credential entries exist on all suppliers, the next step is to perform initial synchronization among all the servers. In our example, most of the initial configuration for all suppliers was done on one server. Thus, the **ds2ldif** command can be used to unload all of the directory data from that server into an LDIF file. The LDIF can be transported through FTP to all the systems where LDAP server instances will be deployed. Then the **ldapadd** or **ldif2ds** (if TDBM) command can be used to prime all other servers in the topology. After **ldapadd** or **ldif2ds** is complete, the configuration is complete. The console messages in Figure 6-7 should be seen on all supplier systems.

```

GLD8650I Replication agreement
'CN=AGGR1,CN=MASTER,IBM-REPLICAGROUP=DEFAULT,0=OLYMPIA,0=SAMPLELDBM' is active.

GLD8517I Replication starting for replica
'CN=AGGR1,CN=MASTER,IBM-REPLICAGROUP=DEFAULT,0=OLYMPIA,0=SAMPLELDBM'.

GLD8650I Replication agreement
'CN=AGGR1,CN=MASTER,IBM-REPLICAGROUP=DEFAULT,0=OLYMPIA,0=SAMPLETDBM' is active.

GLD8517I Replication starting for replica
'CN=AGGR1,CN=MASTER,IBM-REPLICAGROUP=DEFAULT,0=OLYMPIA,0=SAMPLETDBM'.

```

Figure 6-7 Console messages

At this point, the servers are ready to actively participate in replication. Maintenance mode can be turned off and clients can begin sending updates.

6.5.4 Maintaining the Topology

IBM Tivoli Directory Server for z/OS provides many tools to maintain an active replication topology. First, certain replication agreement operational attributes can be retrieved to determine the state of supplier to consumer replication queue as shown in Table 6-1.

Table 6-1 Replication agreement operational attributes

Operational Attributes	Description
ibm-replicationChangeLdif	This attribute provides the pending changes in an LDIF format. Pending changes are the updates in the context that are to be replicated to the consumer, but are still pending in the queue.
ibm-replicationLastResult	This attribute provides the results of the last attempted update, in the form: timestamp changeid resultcode operation entry DN
ibm-replicationPendingChangeCount	This attribute indicates the number of changes pending on an agreement.
ibm-replicationNextTime	This attribute returns the time when the next pending change will be dispatched by this agreement to the consumer. This value is useful when using scheduling.
ibm-replicationState	This attribute indicates the state of the agreement. The following are the values that ibm-replicationState can take: <ul style="list-style-type: none"> ▶ Active means that replication is going on over this agreement. ▶ Binding indicates that the supplier is in the process of binding to the consumer ▶ On Hold indicate that the supplier replication agreement's replication processing is on hold. ▶ Retrying indicates that the supplier is retrying a queued reapplication update ▶ Waiting indicates that the agreement is waiting for agreement to be activated ▶ Connecting indicates that the agreement is currently waiting for the supplier to connect to the consumer. ▶ Suspended indicates that the agreement is suspended and no more replication updates will be sent to the consumer by this agreement until it returns to the ready state ▶ Full indicates that the queue for this agreement is full, and also displays a value that indicates the amount of progress. ▶ Ready indicates immediate replication mode, ready to send updates as they occur.
ibm-replicationLastResultAdditional	This attribute returns the message component from the last attempted update.
Ibm-replicationLastFinishTime	This attribute returns the time when the last pending change was dispatched by this agreement to the consumer.
Ibm-replicationLastActivationTime	This attribute returns the time that the last replication session started between this supplier and consumer.

Operational Attributes	Description
IBM-replicationPendingChanges	This attribute provides the pending changes, one line per change, including the change ID and the operation.
ibm-replicationLastChangeId	This attribute indicates the change ID of the last completed change sent to the consumer.
ibm-replicationFailedChangeCount	This attribute indicates the number of changes that resulted in an error being returned by the consumer.
IBM-replicationFailedChanges	This attribute provides the changes that resulted in an error, one line per error, including the change ID and the operation.
ibm-replicationOnHold	This attribute indicates if the agreement is on hold, i.e., changes are queued, but not replicated to consumer.

Similarly, replication context entries have operational attributes that provide information as well as shown in Table 6-2.

Table 6-2 Replication context operational attributes

Operational Attribute	Description
ibm-replicationIsQuiesced	This attribute returns the quiesced state of replication context. Quiesced contexts will reject non-administrator client updates.
ibm-replicationThisServerIsMaster	This attribute returns true if the server is a master for the context, i.e., a <code>ibm-replicaSubentry</code> contains the server's server ID and its <code>is-master</code> attribute is set to true.

A simple shell script can be used to retrieve all the attributes for all replication related entries in the directory is shown in Figure 6-8.

```

ldapsearch -h localhost -D cn=admin -w secret -p 389 -s sub -b
"o=olympia,o=sampleTDBM" "objectclass=ibm-replication*" dn \
ibm-replicationChangeLdif \
ibm-replicationLastResult \
ibm-replicationPendingChangeCount \
ibm-replicationNextTime \
ibm-replicationState \
ibm-replicationLastResultAdditional \
ibm-replicationPerformance \
ibm-replicationLastFinishTime \
ibm-replicationLastActivationTime \
ibm-replicationPendingChanges \
ibm-replicationLastChangeId \
ibm-replicationFailedChangeCount \
ibm-replicationFailedChanges \
ibm-replicationOnHold \
ibm-replicationIsQuiesced ibm-replicationThisServerIsMaster

```

Figure 6-8 Shell script to retrieve replication attributes

The operational attributes provide state information that might show the queue requires maintenance. IBM Tivoli Directory Server for z/OS provides various extended operations to manage advanced replication. Table 6-3 lists all of the extended operations and the operational attributes that will provide information necessary for using the extended operation.

Table 6-3 Advanced replication related extended operations

Extended Operation	Idapexop -op	Operational Attribute	Description
Cascading Control Replication extop	cascrepl	ibm-replicationState ibm-replicationIsQuiesced	This extended operation manages the state of the replication queue when the supplier is sending updates to a consumer that is also a supplier (a cascading server). It allows users to quiesce and unquiesce the supplier and wait until that operation has been applied to all consumers in the topology. It also allows the user to force replication on the supplier and wait until the queue has been completely replicated to all of the consumers in the topology.
Control Replication	controlrepl	ibm-replicationState	This extended operation manages the state of the replication queue. It allows the user to suspend, resume, or immediately kick off replication (in the case of scheduled replication).
Control Replication Queue	controlqueue	ibm-replicationPendingChanges ibm-replicationPendingChangeCount	This extended operation manages the pending changes in the queue. It allows the user to skip pending operations.
Control Replication Error Log	controlreple rr	ibm-replicationFailedChanges ibm-replicationFailedChangeCount	This extended operation manages the error log. It allows the user to delete the error, show the LDIF of the update causing the error, and retry the operation that caused the error.
Quiesce Context	quiesce	ibm-replicationIsQuiesced	This extended operation quiesces or unquiesces a replication context. Similar to maintenance mode, a quiesced context cannot receive updates from a client unless the client is an administrator and the update includes the administrator control.
Replicate Topology Entries	repltopology	n/a	This extended operation ensures the LDAP entries necessary to set up a supplier are replicated to all the consumers. This extended operation is commonly used when you are setting up a directory that will be replicated while it is being populated.

IBM Tivoli Directory Server ships with the **Idapexop** command, a UNIX Systems Services Shell command line utility that can be used to issue the extended operations shown in Table 6-3.

6.6 Additional Advanced Replication Features

As outlined throughout this section, IBM Tivoli Directory Server for z/OS's advanced replication provides a great amount of flexibility. Sub-tree based replication, a variety of topologies to choose from, operational attributes to monitor status, and extended operations to manage replication are the main features discussed thus far. However there is more. Advanced replication provides additional opportunity for customization of the replication environment.

6.6.1 Scheduling

One feature that resembles a familiar paradigm seen on mainframes is scheduled replication. Scheduled replication allows for pending changes to be queued and then replicated at designated times. To set up scheduled replication, the replication agreement objects must contain a value for the **ibm-replicascheduledn** attribute. This value must define the distinguished name of an entry with the **ibm-replicationDailySchedule** or **ibm-replicationWeeklySchedule** object class.

In the case of daily scheduling, a time for immediate replication can be set using the **ibm-replicationImmediateStart** attribute in an **ibm-replicationDailySchedule** entry. This attribute contains the time in 24 hour format to begin replicating updates. It will continue replicating updates as they arrive until a batch start time is defined. A batch start is defined by using the **ibm-replicationBatchStart** attribute. The **ibm-replicationBatchStart** attribute defines the time when all the pending changes are replicated. Unlike the immediate start, batch start will result in any new changes after the start time being queued until the next immediate or batch start. An example is shown in Figure 6-9.

```
dn: cn=tight, cn=localhost
objectclass: ibm-replicationDailySchedule
ibm-replicationTimesUTC: FALSE
ibm-replicationBatchStart: T183500
ibm-replicationBatchStart: T183700
ibm-replicationImmediateStart: T183600
ibm-replicationImmediateStart: T183600
ibm-replicationImmediateStart: T184000
```

Figure 6-9 **ibm-replicationBatchStart**

For weekly scheduling, the **ibm-replicationWeeklySchedule** entry contains attributes for each day in the week:

- ▶ **ibm-scheduleSunday**
- ▶ **ibm-scheduleMonday**
- ▶ **ibm-scheduleTuesday**
- ▶ **ibm-scheduleWednesday**
- ▶ **ibm-scheduleThursday**
- ▶ **ibm-scheduleFriday**
- ▶ **ibm-scheduleSaturday**

These attributes define the distinguished names of entries of type **ibm-replicationDailySchedule** or **ibm-replicationWeeklySchedule**. An example is shown in Figure 6-10 on page 154.

```

dn: cn=tight schedule, cn=localhost
objectclass: ibm-replicationWeeklySchedule
ibm-scheduleSunday: cn=tight, cn=localhost
ibm-scheduleMonday: cn=tight, cn=localhost
ibm-scheduleTuesday: cn=tight, cn=localhost
ibm-scheduleWednesday: cn=tight, cn=localhost
ibm-scheduleThursday: cn=tight, cn=localhost
ibm-scheduleFriday: cn=tight, cn=localhost
ibm-scheduleSaturday: cn=tight, cn=localhost

```

Figure 6-10 Weekly and Daily scheduling

6.6.2 Filtering

Another feature of advanced replication is the option to partially replicate updates. This is done by introducing a filter that is applied against all replicated updates prior to sending the update. This filter would remove any portions of the update prior to replicating to the consumer. Similar to scheduling, a filter must be associated with the replication agreement object using the **ibm-replicationfilterdn** attribute. This attribute must define the distinguished name of an entry with **ibm-replicationFilter** object class. This entry contains one or more values for the **ibm-replicationFilterAttr** attribute that represent the actual filter(s). The basic syntax of a filter is:

```

(objectclass=objectclassOfEntryToFilter):(attr1toInclude,attr2ToInclude,...)
(objectclass=objectclassOfEntryToFilter):!(attr1ToRemove,attr2ToRemove,...)

```

Here are examples:

Only replicate cn, sn, userpassword of person entries:

```
(objectclass=person):(cn,sn,userpassword)
```

Replicate all attributes of a person entry except the telephone number and employee number:

```
(objectclass=person):!(telephonenumber, employeeNumber)
```

Wild-cards are supported as shown in this example to replicate all attributes of all entries:

```
(objectclass=*)(*)
```

Conceivably filters can be defined that preclude an entire update from being replicated. If the update is an LDAP add to a parent entry, all replicated add operations for children of that parent entry would fail due to the missing parent. To address this, the replication agreement can be updated to include the **ibm-replicationCreateMissingEntries** attribute with a TRUE value. If this attribute is set to TRUE and a missing parent error is encountered, IBM Tivoli Directory Server will send a consumer a generated parent entry that will resolve all missing parent errors.



Plug-ins

This chapter provides information about IBM Tivoli Directory Server plug-ins.

7.1 IBM Tivoli Directory Server for z/OS Server Plug-ins

The functionality of IBM Tivoli Directory Server for z/OS can be extended by software plug-ins. Plug-in support enables the development of routines that are plugged in to the server to handle client request processing.

The following types of plug-ins are supported by IBM Tivoli Directory Server for z/OS:

- ▶ Pre-operation: A plug-in that is executed before a client request is processed.
- ▶ Client-operation: A plug-in that is called to process a client request.
- ▶ Post-operation: A plug-in that is executed after a client request is processed.

After a plug-in has been developed, it is subsequently built into a dynamically loaded library (DLL) that is loaded into the LDAP server's address space when the server is started. When a plug-in is loaded, its designated plug-in initialization routine is called to register its plug-in functions. After the plug-in functions are registered, the server is able to call the plug-in functions for client request processing.

The LDAP server retrieves the plug-in DLL and initialization routine name from the plug-in configuration option. This configuration option can be specified separately in the LDAP configuration file for each of the pre-operation, client-operation, and post-operation plug-ins.

When the LDAP server receives a client request for an ADD, BIND, COMPARE, DELETE, EXTENDED OPERATION, MODIFY, MODIFY DN, or SEARCH, the server processes the request as follows:

1. The server calls all registered pre-operation plug-ins. If a pre-operation plug-in returns a non-zero return code, processing goes to step 4 (skipping steps 2 and 3).
2. If a configured database back end is found that accepts the client request, that back end processes the request.
3. If a client request is not accepted by a configured database back end, then if a registered client-operation plug-in is found that accepts the client request, that plug-in processes the request.
4. The server calls all registered post-operation plug-ins.

When a request is processed by a configured database back end or by a client-operation plug-in (steps 2 and 3), that back end or plug-in must return a message to the client. If the client request is not processed, the LDAP server returns an error message to the client. Only one message is returned to the client.

Figure 7-1 shows the server flow for the request processing described in steps 1-4.

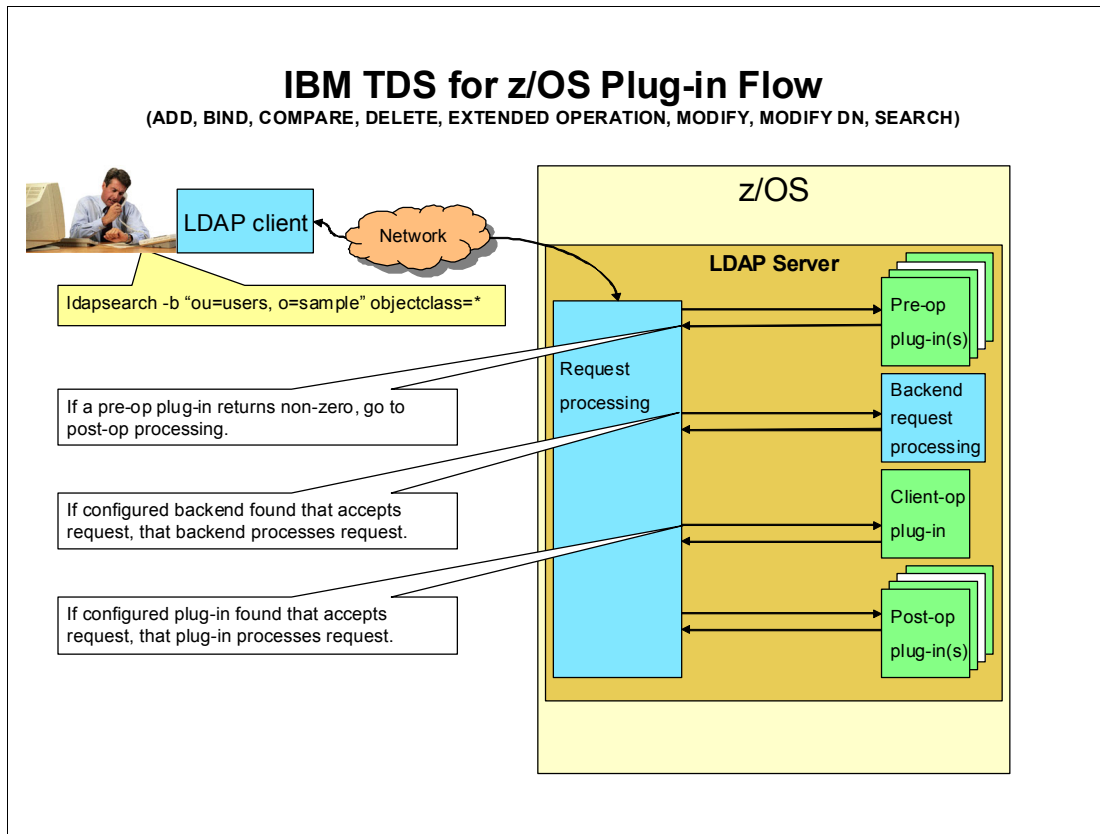


Figure 7-1 IBM Tivoli Directory Server for z/OS Plug-in Flow

When the LDAP server receives a client request for an ABANDON or UNBIND, the server processes the request as follows:

1. The server first calls all registered pre-operation plug-ins.
2. The server processes the request.
3. The server calls all registered client-operation plug-ins.
4. The server then calls all registered post-operation plug-ins.

Figure 7-2 shows the server flow for the ABANDON and UNBIND request processing described in steps 1-4.

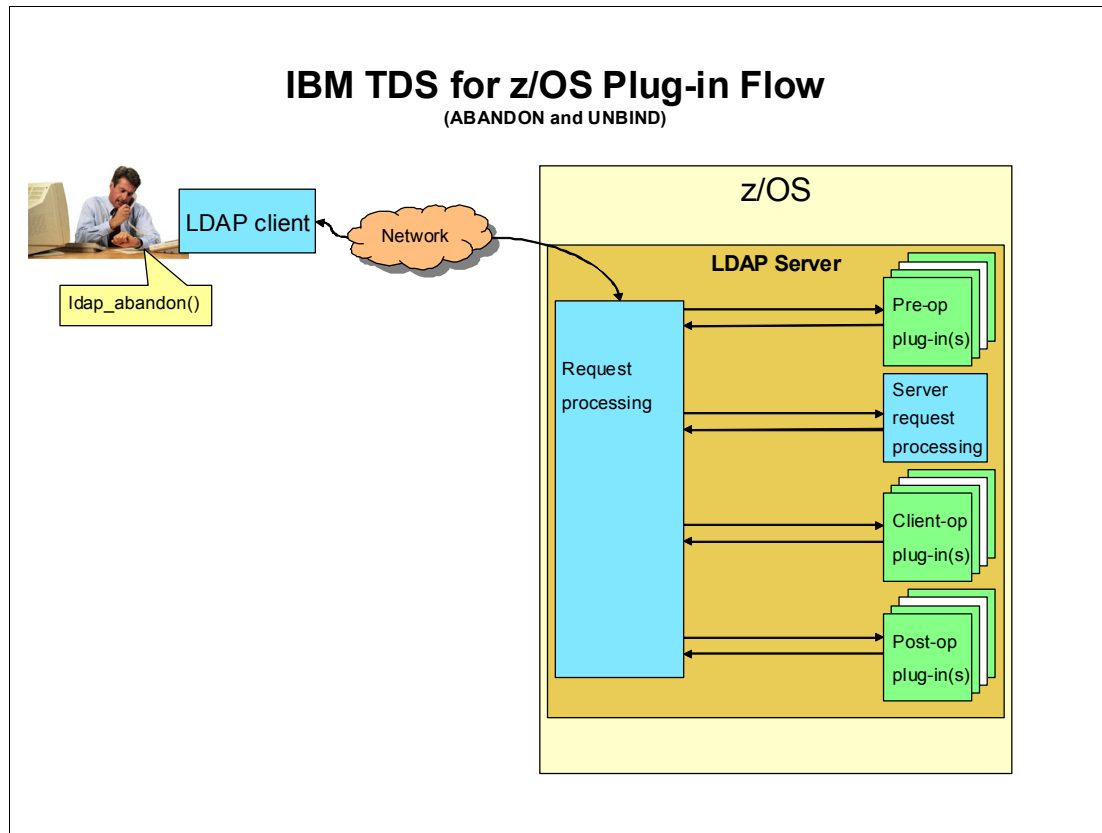


Figure 7-2 IBM Tivoli Directory Server for z/OS Plug-in Flow (Abandon and Unbind)

When an ABANDON or UNBIND request is processed by a configured database back end or by a client-operation plug-in, the back-end or plug-in does not return a response to the client because there is no client response for these type of requests.

This chapter gives an overview of the IBM Tivoli Directory Server for z/OS plug-in support. For more details about the IBM Tivoli Directory Server for z/OS plug-in support and a detailed reference describing each of the SLAPI plug-in application service routines (routines that are prefixed with `slapi_`), see *V1R10.0 IBM Tivoli Directory Server Plug-in Reference for z/OS*, SA76-0148-00.

7.2 Pre-operation and post-operation plug-ins

A pre-operation plug-in is executed before a client request is processed and a post-operation plug-in is executed after a client request is processed.

Pre-operation and post-operation plug-ins can be written for a variety of reasons. One reason for writing a pre-operation plug-in is to check for the existence of a new entry before the new entry is added to a directory. One reason for writing a post-operation plug-in is to audit clients after they bind to the server.

The plug-in initialization function is responsible for registering the functions for request message types supported by the plug-in by calling the `slapi_pblock_set()` routine. The plug-in will not be called for a request message type that it has not registered.

The prototype for a pre-operation plug-in must return an integer and take a plug-in parameter block (`Slapi_PBlock`) pointer as the input parameter. For example:

```
int pre_op_function ( Slapi_PBlock * pb );
```

The return value from the pre-operation function is zero if request processing continues and non-zero if request processing terminates. If a non-zero value is returned, the pre-operation plug-in must return a result message to the client by calling the `slapi_send_ldap_result()` routine. If a zero value is returned, the pre-operation plug-in must not return a result to the client. A result message is not returned for ABANDON and UNBIND requests, and the plug-in return value is ignored for these message types. Post-operation plug-ins are called even if a nonzero value is returned by the pre-operation plug-in.

The prototype for a post-operation plug-in must return a void and take a plug-in parameter block pointer as the parameter. For example:

```
void post_op_function ( Slapi_PBlock * pb );
```

A post-operation function does not return a value. A post-operation plug-in must not return a result message to the client because this will have already been done before the post-operation plug-in is called. The `slapi_pblock_get()` routine can be called to obtain the result code returned to the client for the request.

Pre-operation and post-operation plug-in functions can be registered by the `slapi_pblock_set()` routine to handle pre and post client requests for any of the following request message types:

ABANDON	Each pre-operation and post-operation plug-in is called for an ABANDON request if the plug-in has registered corresponding <code>SLAPI_PLUGIN_ABANDON_FN</code> routine(s).
ADD	Each pre-operation and post-operation plug-in is called for an ADD request if the plug-in has registered corresponding <code>SLAPI_PLUGIN_ADD_FN</code> routine(s).
BIND	Each pre-operation and post-operation plug-in is called for a BIND request if the plug-in has registered corresponding <code>SLAPI_PLUGIN_BIND_FN</code> routine(s).
COMPARE	Each pre-operation and post-operation plug-in is called for a COMPARE request if the plug-in has registered corresponding <code>SLAPI_PLUGIN_COMPARE_FN</code> routine(s).
DELETE	Each pre-operation and post-operation plug-in is called for a DELETE request if the plug-in has registered corresponding <code>SLAPI_PLUGIN_DELETE_FN</code> routine(s).
EXTENDED OPERATION	Each pre-operation and post-operation plug-in is called for an EXTENDED OPERATION request if the plug-in has registered corresponding <code>SLAPI_PLUGIN_EXT_OP_FN</code> routine(s).
MODIFY	Each pre-operation and post-operation plug-in is called for a MODIFY request if the plug-in has registered corresponding <code>SLAPI_PLUGIN_MODIFY_FN</code> routine(s).

MODIFY DN	Each pre-operation and post-operation plug-in is called for a MODIFY DN request if the plug-in has registered corresponding SLAPI_PLUGIN_MODRDN_FN routine(s).
SEARCH	Each pre-operation and post-operation plug-in is called for a SEARCH request if the plug-in has registered corresponding SLAPI_PLUGIN_SEARCH_FN routine(s).
UNBIND	Each pre-operation and post-operation plug-in is called for an UNBIND request if the plug-in has registered corresponding SLAPI_PLUGIN_UNBIND_FN routine(s).

7.3 Client-operation plug-ins

A client-operation plug-in is executed to handle a client request. Client-operation plug-ins can be written for a variety of reasons. One reason for writing a client-operation plug-in could be to implement a back end directory that is independent from any of those provided by IBM Tivoli Directory Server for z/OS.

For ADD, BIND, COMPARE, DELETE, MODIFY, MODIFY DN and SEARCH requests, the corresponding client-operation plug-in is called if the plug-in registered a suffix that matches the target DN for the request. For EXTENDED OPERATION requests, the corresponding plug-in is called if the plug-in registered an object identifier (OID) that matches the OID specified in the request. All corresponding client-operation plug-ins are called for ABANDON and UNBIND requests.

The client-operation plug-in initialization function is responsible for registering the request message types, distinguished name suffixes and extended operations supported by the plug-in by calling the `slapi_pblock_set()` routine. A plug-in is only called for request message types or extended operations that it has registered for.

The prototype for a client-operation plug-in must return a void and take a plug-in parameter block pointer as the parameter. For example:

```
void client_op_function ( Slapi_PBlock * pb );
```

The client-operation function does not return a value. The client operation plug-in must return a result message to the client for all message types except ABANDON and UNBIND because these message types do not return a response to the client. The `slapi_send_ldap_result()` routine is used to send the result message to the client.

Client-operation plug-in functions can be registered by the `slapi_pblock_set()` routine to handle client requests for any of the following request message types:

ABANDON	Each client-operation plug-in is called for an ABANDON request if the plug-in has registered a SLAPI_PLUGIN_ABANDON_FN routine. The plug-in must not return a response to the client because there is no client response for an ABANDON request. The plug-in stops processing a request that is abandoned by the client.
ADD	The client-operation plug-in is called for an ADD request if the entry DN matches a suffix registered by the plug-in and the plug-in registered a SLAPI_PLUGIN_ADD_FN routine. The plug-in is responsible for processing the request and returning the result message to the client.
BIND	The client-operation plug-in is called for a simple BIND if the authentication DN matches a suffix registered by the plug-in and the

	<p>plug-in registered a <code>SLAPI_PLUGIN_BIND_FN</code> routine. A SASL BIND is not passed to the plug-in. The plug-in is responsible for authenticating the DN and returning the result message to the client. Extended group gathering is performed for an authentication DN located in a plug-in database but plug-in databases are not included in the group gathering process.</p>
COMPARE	<p>The client-operation plug-in is called for a COMPARE request if the entry DN matches a suffix registered by the plug-in and the plug-in registered a <code>SLAPI_PLUGIN_COMPARE_FN</code> routine. The plug-in is responsible for processing the request and returning the result message to the client.</p>
DELETE	<p>The client-operation plug-in is called for a DELETE request if the entry DN matches a suffix registered by the plug-in and the plug-in registered a <code>SLAPI_PLUGIN_DELETE_FN</code> routine. The plug-in is responsible for processing the request and returning the result message to the client.</p>
EXTENDED OPERATION	<p>The client-operation plug-in is called for an EXTENDED OPERATION request if the request OID matches an OID registered by the plug-in and the plug-in registered a <code>SLAPI_PLUGIN_EXT_OP_FN</code> routine. The plug-in is responsible for processing the extended operation request and returning the result to the client. The <code>slapi_pblock_set()</code> routine is used to set the extended operation result OID (<code>SLAPI_EXT_OP_RET_OID</code>) and value (<code>SLAPI_EXT_OP_RET_VALUE</code>) in the result message. The <code>slapi_send_ldap_result()</code> routine is then used to return the result to the client.</p>
MODIFY	<p>The client-operation plug-in is called for a MODIFY request if the entry DN matches a suffix registered by the plug-in and the plug-in registered a <code>SLAPI_PLUGIN_MODIFY_FN</code> routine. The plug-in is responsible for processing the request and returning the result message to the client.</p>
MODIFY DN	<p>The client-operation plug-in is called for a MODIFY DN request if the entry DN matches a suffix registered by the plug-in and the plug-in registered a <code>SLAPI_PLUGIN_MODRDN_FN</code> routine. The plug-in is responsible for processing the request and returning the result message to the client.</p>
SEARCH	<p>The client-operation plug-in is called for a SEARCH request if the base DN matches a suffix registered by the plug-in and the plug-in registered a <code>SLAPI_PLUGIN_SEARCH_FN</code> routine. The plug-in is responsible for processing the request and returning the result message to the client. Search entries are returned by calling the <code>slapi_send_ldap_search_entry()</code> routine, search referrals are returned by calling the <code>slapi_send_ldap_referral()</code> routine, and the search result is then returned by calling the <code>slapi_send_ldap_result()</code> routine.</p>
UNBIND	<p>Each client-operation plug-in is called for an UNBIND request if the plug-in registered a <code>SLAPI_PLUGIN_UNBIND_FN</code> routine. The plug-in must not return a response to the client because there is no client response for an UNBIND request. The plug-in does not release any resources that are allocated for the connection.</p>

7.4 Building an IBM Tivoli Directory Server for z/OS server plug-in

Each plug-in is a separate DLL that is loaded by the LDAP server. The `slapi-plugin.h` include file defines the various structures and service routine prototypes that are available to the plug-in.

LDAP server SLAPI export definitions are contained in one of two DLL load modules:

- ▶ The `GLDSLP31.x` side file contains the export definitions for a 31-bit plug-in DLL.
- ▶ The `GLDSLP64.x` side file contains the export definitions for a 64-bit plug-in DLL.

The plug-in must be stored as a member of a PDS or PDSE. In addition, a 64-bit plug-in DLL must be stored in a PDSE. The plug-in data set must be in the load list for the LDAP server, either through a `STEPLIB` statement or the system `LNKLST`.

The LDAP server `plugin` configuration option is used to define a plug-in, and must be added to the LDAP server configuration file. It has three required parameters and one optional parameter:

1. The plug-in type: `preOperation`, `clientOperation` or `postOperation`
2. The plug-in DLL name
3. The name of the plug-in initialization routine that will be called during LDAP server initialization
4. Optional parameters that the plug-in can retrieve

For example:

```
plugin postOperation PLUGSAMP plugin_init "auditFile"
```

7.5 Steps for writing a IBM Tivoli Directory Server for z/OS server plug-in

To build an IBM Tivoli Directory Server for z/OS plug-in, perform the following steps:

1. Design and write the plug-in initialization routine and SLAPI service functions

The plug-in initialization routine must register the following that are supported by the plug-in:

- Request message functions
- Service functions
- Distinguished name suffixes
- Extended operation OIDs

Return code 0 must be returned when successful and non-zero when not successful. The plug-in initialization routine receives as input, the plug-in parameter block (`Slapi_PBlock`) and returns an integer as the return value. An example of an initialization routine prototype:

```
int plugin_init ( Slapi_PBlock * pb );
```

2. When writing the SLAPI service functions that implement the plug-in design, see *V1R10.0 IBM Tivoli Directory Server Plug-in Reference for z/OS*, SA76-0148-00 for application

service routines to use and for defined prototypes. You can also see `slapi-plugin.h` for defined prototypes.

3. Decide on any input parameters for the plug-in. Plug-in input parameters can be retrieved using the `SLAPI_PLUGIN_ARGC` or `SLAPI_PLUGIN_ARGV` parameters with the `slapi_pblock_get()` service routine.
4. Include `slapi-plugin.h`, which contains defined SLAPI data structures and prototypes.
5. Export the plug-in initialization routine.
6. Compile the plug-in code into object files.
7. Link the plug-in object files with one of the LDAP server SLAPI side files listed above.
8. Ensure the plug-in DLL module is in the load list of the LDAP server and is a member of either a PDS or PDSE.
9. APF authorize the data set that contains the plug-in DLL.
10. Edit and add the `plugin` configuration option to the LDAP server configuration file.
11. Restart the LDAP server

You might want to program trace statements to follow processing flow in the plug-in. The trace macro, `SLAPI_TRACE()`, is provided in `slapi-plugin.h`. The syntax is:

```
SLAPI_TRACE((LDAP_DEBUG_PLUGIN, "PLUGSAMP", "Entered."));
```

7.6 IBM Tivoli Directory Server for z/OS Server Plug-in Sample

A sample plug-in and its makefile are provided in `/usr/lpp/ldap/examples`.

The sample plug-in, `/usr/lpp/ldap/examples/plugin_sample.c`, creates a post-operation plug-in that logs LDAP server BIND requests and result codes to a log file. The log file is specified as an input parameter on the `plugin` configuration option.

For reference, `plugin_sample.c` is also located in Appendix A, "Sample plug-in code" on page 293.

The makefile, `/usr/lpp/ldap/examples/makefile.plugin`, is used for building `plugin_sample.c`.

7.6.1 Stepping through `plugin_sample.c`

The initialization routine, client request BIND routine, CLOSE routine, and function calls in `plugin_sample.c` help to give a better understanding of plug-in coding practices and code flows. The following bullets each describe a line of code from `plugin_sample.c`, followed by the actual line number and line of code. For more details about any of the SLAPI macros or functions described here, see *V1R10.0 IBM Tivoli Directory Server Plug-in Reference for z/OS*, SA76-0148-00.

- ▶ The `SLAPI_TRACE` macro uses the `slapi_trace()` service routine to write an LDAP server trace message. Note that the `LDAP_DEBUG_PLUGIN` trace messages are only issued if the `PLUGIN` debug level was requested.

```
90     SLAPI_TRACE((LDAP_DEBUG_PLUGIN, "PLUGSAMP", "Entered."));
```

- ▶ The `SLAPI_PLUGIN_TYPE` operational parameter is specified on a call to the `slapi_pblock_get()` routine to determine whether the plug-in initialization routine was called for a pre-operation, client-operation, or post-operation plug-in. In this case, if the

initialization routine was not called for a post-operation plug-in, then the initialization routine must have incorrectly been specified for a pre-operation or client-operation on the **plugin** configuration file option.

```
95 rc = slapi_pblock_get(pb, SLAPI_PLUGIN_TYPE, &type);
```

- ▶ The **slapi_log_error()** routine is used to write a message to the LDAP server job log, and optionally to the console.

```
97 slapi_log_error(LDAP_MSG_HIGH, "PLUGSAMP",  
98 "Unable to get plug-in type: Error %d\n",errno);
```

- ▶ The SLAPI_PLUGIN_ARGC operational parameter is specified on a call to the **slapi_pblock_get()** routine to retrieve the number of arguments that were specified on the **plugin** configuration statement.

```
111 rc = slapi_pblock_get(pb, SLAPI_PLUGIN_ARGC, &argc);
```

- ▶ The SLAPI_PLUGIN_ARGV operational parameter is specified on a call to the **slapi_pblock_get()** routine to retrieve a NULL-terminated array of arguments that were specified on the **plugin** configuration statement.

```
127 rc = slapi_pblock_get(pb, SLAPI_PLUGIN_ARGV, &argv);
```

- ▶ The SLAPI_PLUGIN_CLOSE_FN registration parameter is specified on a call to the **slapi_pblock_set()** routine to register the **plugin_close_fn()** routine to be called during LDAP server termination.

```
138 rc = slapi_pblock_set(pb, SLAPI_PLUGIN_CLOSE_FN, (void *)plugin_close_fn);
```

- ▶ The SLAPI_PLUGIN_BIND_FN registration parameter is specified on a call to the **slapi_pblock_set()** routine to register the **plugin_bind_fn()** routine to process a client BIND request.

```
148 rc = slapi_pblock_set(pb, SLAPI_PLUGIN_BIND_FN, (void *)plugin_bind_fn);
```

- ▶ The **slapi_ch_malloc()** routine is used to allocate storage for use by the plug-in. The **slapi_ch_free()** routine is used to release the storage when it is no longer needed.

```
158 pdata = (plugin_private *)slapi_ch_malloc(sizeof(plugin_private));
```

- ▶ The **fopen()** C/C++ API is used to open the log file, the name of which was specified using the **plugin** configuration option.

```
218 pdata->auditFile = fopen(pdata->auditFilename, "a");
```

- ▶ The SLAPI_PLUGIN_PRIVATE operational parameter is specified on a call to the **slapi_pblock_set()** routine to store the pointer to the storage where the initialized **plugin_private** structure resides.

```
229 rc = slapi_pblock_set(pb, SLAPI_PLUGIN_PRIVATE, &pdata);
```

- ▶ The SLAPI_PLUGIN_PRIVATE operational parameter is specified on a call to the **slapi_pblock_get()** routine to retrieve the pointer to the storage where the initialized **plugin_private** structure resides.

```
281 rc = slapi_pblock_get(pb, SLAPI_PLUGIN_PRIVATE, &pdata);
```

- ▶ The SLAPI_BIND_TARGET bind request parameter is specified on a call to the **slapi_pblock_get()** routine to retrieve the authentication DN from the BIND request.

```
302 rc = slapi_pblock_get(pb, SLAPI_BIND_TARGET, &bindDN);
```

- ▶ The SLAPI_PLUGIN_OPRETURN general result parameter is specified on a call to the **slapi_pblock_get()** routine to retrieve the result code for the current operation. The result code can be set by the **slapi_send_ldap_result()** routine. In the case of this sample, a configured back end database routine has already set the result code.

```
309 rc = slapi_pblock_get(pb, SLAPI_PLUGIN_OPRETURN, &resultCode);
```

- ▶ The `SLAPI_REQUESTOR_GROUPS` general request parameter is specified on a call to the `slapi_pblock_get()` routine to retrieve a NULL-terminated array of normalized group names for the authentication DN. The value is NULL if the authentication DN is not a member of any groups or if group gathering was not enabled for the BIND request.

```
316 rc = slapi_pblock_get(pb, SLAPI_REQUESTOR_GROUPS, &groupList);
```

- ▶ The `SLAPI_REQUESTOR_ALT_NAMES` general request parameter is specified on a call to the `slapi_pblock_get()` routine to retrieve a NULL-terminated array of normalized alternate names for the authentication DN. The value is NULL if there are no alternate names.

```
323 rc = slapi_pblock_get(pb, SLAPI_REQUESTOR_ALT_NAMES, &dnList);
```

- ▶ The `fprintf()` C/C++ API is used to write a log record to the log file, the name of which was specified on the `plugin` configuration option.

```
346 rc = fprintf(pdata->auditFile, "Result: %d DN: %s\n", resultCode, cnvName);
```

7.6.2 Steps for building and running the sample plug-in

To build and run the sample plug-in, perform the following steps:

1. Start by creating either a PDS or a PDSE dataset with the same attributes as `SYS1.SIEALNKE`. A PDSE dataset is required when building the plug-in sample as a 64-bit module.
2. APF authorize the dataset created.
3. Ensure the dataset is in the load list for the LDAP server, either through a `STEPLIB` statement or the system `LNKLST`.
4. Copy `/usr/lpp/ldap/examples/plugin_sample.c` and `/usr/lpp/ldap/examples/makefile.plugin` to a directory that you have write access to.
5. Edit `makefile.plugin` and update `PLUGSAMP_DLL` with the name of the dataset you created. For example:

```
PLUGSAMP_DLL = '//'GLD.PLUGIN.SIEALNKE(PLUGSAMP)''
```

Also, if you are building a 64-bit DLL, set `PLUGSAMP_ADDR_MODE` to 64.

6. Save `makefile.plugin`.
7. To compile and linkedit the sample plug-in using the `makefile.plugin`, enter **`make -f makefile.plugin`**.
8. Verify that no build or link errors occurred. Verify that your dataset now contains the member `PLUGSAMP`, or a member with the name you updated.
9. Stop the server.
10. Edit the LDAP server configuration file and add the `plugin` configuration option to the global section:

```
plugin postOperation PLUGSAMP plugin_init "logFilename"
```

"logFilename" is the name of the file you want to have the log records written to, and it must be in double quotes.

11. If you are building a 64-bit DLL, then add the `plugin` configuration option in the following format:

```
plugin postOperation PLUGSM31/PLUGSAMP plugin_init "logFilename"
```

A plug-in that supports both 31-bit and 64-bit addressing modes should specify both file names separated by a slash (/), such as `plugin31/plugin64`. For this 64-bit example,

PLUGSAMP is the name used when the 64-bit DLL was built, as shown above. Because the syntax of the configuration option requires a 31 bit plug-in name preceding the / when specifying a 64 bit plug in DLL, the name PLUGSM31 was used as placeholder name for the **plugin** configuration option. Because in this example we do not create a 31 bit version, this can be any name, and no DLL with that name needs to exist.

12. Restart the LDAP server.

If you use the debug parameter PLUGIN, sample plug-in trace messages will be written to the LDAP server job log. For example:

```
START LDAPSRV,PARMS=' -d PLUGIN '
```

LDAPSRV is an example name and represents the name of your LDAP server start-up procedure.

After it is started, browse your LDAP server job log for plug-in initialization and trace messages. Also, verify that the sample plug-in created an empty log file.

To test, perform an LDAP operation binding to the LDAP server. The sample plug-in will write a message to the log including the result code of the bind operation and the bind DN. For example:

```
Result: 0 DN: o=your company
```

7.7 Exploiters of IBM Tivoli Directory Server for z/OS Plug-in Support

The IBM Tivoli Directory Server for z/OS plug-in functionality is exploited internally by IBM Tivoli Directory Server for z/OS as well as other z/OS deliverables. The IBM Tivoli Directory Server for z/OS Advanced Replication support is implemented with LDAP plug-ins. Additionally, the following z/OS deliverables use LDAP plug-ins:

- ▶ z/OS Integrated Security Services EIM (Enterprise Identity Mapping)

z/OS EIM allows administrators and application developers to more easily manage multiple user registries and user identities.

The ICTX Java API that ships with z/OS EIM provides the ability to use IBM Tivoli Directory Server for z/OS to perform remote RACF authorization and auditing, and to remotely access the z/OS Identity Cache. ICTX is an LDAP extended operation and is implemented as an LDAP client-operation plug-in.

For more information, see *Integrated Security Services Enterprise Identity Mapping (EIM) Guide and Reference*, SA22-7875.

- ▶ z/OS HCD (Hardware Configuration Definition)

z/OS HCD provides an interactive interface that allows the definition of hardware configuration for both a processor's channel subsystem and the operating system running on the processor, and stores the entire configuration data in a central repository: the input/output definition file (IODF).

Together with IBM Tivoli Directory Server for z/OS and the RACF back end SDBM, the HCD LDAP back end can be used to remotely access and update IODF data through LDAP.

For more information, see *HCD User's Guide*, SC33-7988.



Workload Management

This chapter describes IBM Tivoli Directory Server support for the z/OS Workload Manager.

8.1 Workload Management Overview

The idea of z/OS Workload Manager is to make a contract between IBM Tivoli Directory Server for z/OS and the operating system. IBM Tivoli Directory Server for z/OS allows different kinds of work running on the server to be classified in separate ways so that they can be given different goals in z/OS Workload Manager. This allows the user to set separate goals for each work classification in IBM Tivoli Directory Server for z/OS.

Most work should run under the same goal setting in WLM. However some transactions need to be given separate goals, either at a higher or lower velocity. These transactions should be considered exceptions. Transactions can be differentiated by their client IP address, bind DN (Distinguished Name), or their search pattern.

A search pattern is a collection of search attributes, where arbitrary data is removed, used to identify similar search requests made to IBM Tivoli Directory Server for z/OS. This data is collected by the operations monitor and is useful in conjunction with the IBM Tivoli Directory Server for z/OS WLM support to identify SPAM or performance intensive searches and assign them an appropriate priority in WLM separate from the rest of the IBM Tivoli Directory Server for z/OS work.

All work in IBM Tivoli Directory Server for z/OS normally runs under an enclave with the WLM Transaction Name GENERAL. If the user has created exception classes, these transactions will run under an enclave with the WLM Transaction Name supplied by the user during IBM Tivoli Directory Server for z/OS server configuration.

IBM Tivoli Directory Server for z/OS only supports execution velocity goals for use with Workload Manager. Execution velocity goals define the amount of delay that is acceptable, or, in other words, how long a resource is used compared to the amount of time waiting to use it. Execution velocity depends on how many samples are collected for a service class and therefore, on the amount of work running in a service class. The achievable velocity for a workload also depends on the number of CPUs on the system. For these reasons a default execution velocity is not given for IBM Tivoli Directory Server for z/OS. Look at the RMF™ report for a typical IBM Tivoli Directory Server for z/OS run and determine what the typical execution velocity is for IBM Tivoli Directory Server for z/OS on the user's system. Then determine if this is too high or too low relative to the execution velocities for other work running on the system.

Most IBM Tivoli Directory Server for z/OS transactions should be set to this execution velocity in the service class with WLM Transaction Name GENERAL because most or all IBM Tivoli Directory Server for z/OS work will run under this service class. There should be at least a difference of 10 between execution velocity goal settings in two service classes. Service classes with execution velocity goals that have a difference of less than 10 are almost equivalent in WLM calculations. If you need to create an exception service class for IBM Tivoli Directory Server for z/OS work, set the execution velocity at least 10 lower or higher than the execution velocity for the service class with Transaction Name GENERAL.

8.2 Using Configuration Options

IBM Tivoli Directory Server can exploit WLM to classify work within IBM Tivoli Directory Server based on the client's IP address, the distinguished name (DN) associated with the requests, or both. This allows the installation to set performance goals for requests within IBM Tivoli Directory Server.

For example, for a high priority user, you can set that user's requests to be dispatched by z/OS at a higher priority. Likewise, if there is work coming into the system that is less vital, you can set WLM to classify that work as a lower priority.

WLM Service Classes are used to specify the performance characteristics of requests. Classification rules within WLM assign the incoming request to a service class based on a transaction name. Transaction names are identified in the IBM Tivoli Directory Server configuration file with the `wlmExcept` configuration option.

This configuration will show three transaction types being classified: normal LDAP requests, high priority request (requested to be dispatched at a higher priority), and spam requests (classified with a lower dispatching priority).

The following sections show the configuration options needed in WLM, and the associated changes needed in the configuration file for IBM Tivoli Directory Server.

8.2.1 Configuring WLM to support incoming requests

The following configuration changes are made within the WLM ISPF application panels. The user must have write authority to the WLM policy definition datasets.

Add service classes

Create the service classes necessary to support the LDAP workload. In this example configuration, there are three service classes with the following characteristics:

```
LDAPNORM      Importance 3, execution velocity 50
LDAPHIGH      Importance 3, execution velocity 70
LDAPSPAM      Importance 4, execution velocity 20
```

Figure 8-1 illustrates defining the **LDAPHIGH** service class. The LDAPNORM and LDAPSPAM service classes are defined in much the same way.

```
-----
                                Create a Service Class                                Row 1 to 2 of 2
Command ==> _____

Service Class Name . . . . . LDAPHIGH (Required)
Description . . . . . LDAP workload ex. velocity=70
Workload Name . . . . . WRKLD1 (name or ?)
Base Resource Group . . . . . _____ (name or ?)
Cpu Critical . . . . . NO (YES or NO)

Specify BASE GOAL information. Action Codes: I=Insert new period,
E=Edit period, D=Delete period.

      -- Period -- ----- Goal -----
Action # Duration Imp. Description
-----
   1   _____ 3   _____ Execution velocity of 70
```

Figure 8-1 Creating the WLM service class LDAPHIGH

Adding classification rules

After the service classes are defined, WLM needs to know how to associate incoming work with service classes. This is done with classification rules. In this example configuration, the

service classes that were previously defined are selected based on a transaction name. Transaction name is derived from the LDAP configuration file option `wlmExcept`, or from the `WLMEXCEPT` operator modify command.

Any transactions that arrive in the system with a transaction name of SPAM are assigned to the LDAPSPAM service class, and any transactions that arrive with a transaction name of HIGHPRI are assigned to the LDAPHIGH service class. All other requests are assigned the default service class LDAPNORM, as shown in Figure 8-2.

```

-----
                Modify Rules for the Subsystem Type                Row 1 to 2 of 2
Command ==> _____ Scroll ==> PAGE

Subsystem Type . : LDAP      Fold qualifier names?  Y  (Y or N)
Description . . . LDAP workload

Action codes:  A=After      C=Copy      M=Move      I=Insert rule
               B=Before     D=Delete row R=Repeat  IS=Insert Sub-rule
                                   More ==>
               -----Qualifier-----
Action  Type   Name   Start   Service   Report
-----
_____ 1 TN     SPAM   _____  LDAPNORM  LDAPNORM
_____ 1 TN     HIGHPRI _____ LDAPSPAM  LDAPSPAM
_____ 1 TN     _____ LDAPHIGH  LDAPHIGH
***** BOTTOM OF DATA *****

```

Figure 8-2 Classification rules for LDAP transactions

Save and activate the policy

After creating the service classes and report classes, and setting up the classification rules, save the modified policy to the WLM datasets, and activate the policy for it to take effect.

8.2.2 Configuring LDAP to exploit WLM

LDAP can exploit WLM workload classification by requestor IP address, requestor distinguished name (DN) or a combination of both. This is done using the `wlmExcept` option in the configuration file. The `wlmExcept` configuration option can be specified many times in the configuration file. The order in which they appear in the configuration file determines which option will apply because they are processed on a first come/first served basis.

The `WLMEXCEPT` operator modify command can also be used to change a running LDAP configuration. `WLMEXCEPT` operator modify command specifications are evaluated before any `wlmExcept` configuration options. The `RESET WLMEXCEPT` operator command can be used to remove specific WLM routing that was added using the `WLMEXCEPT` operator command.

You should ensure that WLM transaction names specified on `wlmExcept` configuration options or on the `WLMEXCEPT` operator modify commands be mapped to a WLM service class. If a default service class was specified for all non-mapped transactions, that service class will be chosen. Otherwise, a discretionary service class is selected, resulting in these operations receiving lower priority than other work, which can cause unpredictable results.

DSCONFIG example

After analyzing the `searchStats` and `searchIPStats` attributes returned on a `cn=operations,cn=monitor` search, it has been determined there is a spamming LDAP client

application on IP address 1.2.3.4 that has been affecting performance of the LDAP server. Also, requests from bound user `cn=importantguy,o=ibm` should have a higher priority within the LDAP server.

The LDAP administrator can add the following `wlmExcept` configuration options to route these requests to the appropriate WLM transaction name:

- ▶ `wlmExcept SPAM 1.2.3.4`
- ▶ `wlmExcept HIGHPRI cn=importantguy,o=ibm`

After the LDAP server is restarted, LDAP client requests originating from IP address 1.2.3.4 are routed to WLM transaction name `SPAM` with a service class of `LDAPSPAM`. The server routes requests from bound user `cn=importantguy,o=ibm` to WLM transaction name `HIGHPRI` with a service class of `LDAPHIGH`. All other requests are routed to the default service class of `LDAPNORM`. See *z/OS MVS Planning: Workload Management, SA22-7602* for more information about configuring WLM.

If the transaction name specified on the `wlmExcept` configuration option or on the `WLMEXCEPT` operator modify command does not exist in WLM, any client requests associated with that transaction name would use the default service class `LDAPNORM`. If a default service class is not defined, then client requests are assigned to a discretionary service class.

8.3 Using Workload Manager and Operations Monitor together

IBM Tivoli Directory Server for z/OS can also exploit WLM to classify work within TDS, based on the client's search pattern. The `WLMEXCEPT` operator modify command can be used to change the routing of incoming client requests to new or separate WLM transaction names while the server is running. The `WLMEXCEPT` operator modify command can be used to associate a search pattern in the `cn=operations,cn=monitor` entry to a WLM transaction name. Each time the `WLMEXCEPT` operator modify command is issued, the new mappings are added before any of the configured `wlmExcept` configuration options or previously issued `WLMEXCEPT` operator modify commands.

The `WLMEXCEPT` operator modify commands last for the life of the LDAP server. However, the `RESET WLMEXCEPT` can be issued to remove all previously issued `WLMEXCEPT` operator modify commands and default to using the initial LDAP server configuration. If the operations monitor ID (OPID) is specified on the `RESET WLMEXCEPT` operator modify command, then just that specific WLM routing is removed.

If the operations monitor is enabled, the `searchStats` and `searchIPStats` attributes in the `cn=operations,cn=monitor` entry can be used to identify spamming client applications or certain search requests that should have a higher priority within the LDAP server. This type of information is valuable when configuring LDAP to use WLM transaction names, and assigning service or report classes to those transaction names. For a spamming client application, a WLM transaction name with a low priority service or report class ought to be used. For important search requests, a WLM transaction name with a high priority service or report class ought to be used.

8.4 Workload Manager Health

The IBM Tivoli Directory Server for z/OS uses the WLM health service to indicate a health value to WLM. The WLM health value is used by the TCP/IP sysplex distributor to help route

incoming client requests to servers within the sysplex. After the LDAP server initialization, the WLM health value is set to 100%. The WLM health value is calculated by the number of failures during the past 5000 operations if one minute has passed since the value was last calculated. If the percentage of failures changes by 25% or more, the z/OS LDAP server increases or decreases the WLM health value. An LDAP server operation is considered a failure when it has one of the following return codes:


- LDAP_OPERATIONS_ERROR (1)
- LDAP_TIMELIMIT_EXCEEDED (3)
- LDAP_ADMIN_LIMIT_EXCEEDED (11)
- LDAP_BUSY (51)
- LDAP_UNAVAILABLE (52)
- LDAP_UNWILLING_TO_PERFORM (53)
- LDAP_OTHER (80)



Part 3

Installation and configuration examples

This section provides on installing and configuring IBM Tivoli Directory Server for z/OS.



Implementing IBM Tivoli Directory Server on a single system

This chapter provides a step-by-step procedure for implementing IBM Tivoli Directory Server on z/OS. Different implementations are shown, starting with a basic LDBM. Next, a basic TDBM implementation is shown.

We also will show how to secure the administrator password instead of specifying it in clear text in the configuration file.

9.1 A basic IBM Tivoli Directory Server server with LDBM

The LDBM back end keeps its entries in memory for quick access and requires a minimum amount of setup. When the LDAP server is not running, LDBM stores its directory information in z/OS UNIX System Services files. There is no restriction on the type of information that can be stored in LDBM.

This section describes the steps needed to implement IBM Tivoli Directory Server using LDBM.

The IBM Tivoli Directory Server installation utility **dsconfig** is used to build a PDS containing the jobs, configuration files, and the **ldap**-started task needed to run IBM Tivoli Directory Server.

9.1.1 Prepare the z/OS system

The following steps need to be done prior to implementing IBM Tivoli Directory Server:

- ▶ UNIX System Services must be operating in full-function mode. There must be sufficient space for storing the schema in the file system.
- ▶ IBM Tivoli Directory Server must be installed using the standard SMP/E process.
Refer to *z/OS Program Directory* and *z/OS Distributed File Service zSeries File System Administration* for information about installing IBM Tivoli Directory Server and implementing z/OS UNIX System Services.
- ▶ Workload Manager must be implemented to be able to prioritize IBM Tivoli Directory Server with other work running on the system.
- ▶ **dsconfig** assumes that RACF is the security product in use on the system. If another security product is being used, then the generated job to populate RACF will need to be modified accordingly.
- ▶ Decide on a naming convention for your IBM Tivoli Directory Server servers. The convention used in our examples is LDAPxxyy, where xx is the LPAR ID and yy is a unique instance for this server.

9.1.2 Implementing IBM Tivoli Directory Server with dsconfig

Refer to chapter 5 of *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for more information about each of these steps.

1. Copy the following sample files from `/usr/lpp/ldap/etc` to a new location. You can use `/etc`. However, because we are implementing multiple instances of IBM Tivoli Directory Server on a single system, we chose to create a home directory under `/u` for each instance.
 - `ds.profile`
 - `ds.slapped.profile`
 - `ds.racf.profile`
 - `ds.db2.profile` (not changed in this implementation, however `ds.profile` has a link to this file.)
2. Modify `ds.profile` with the following changes:
 - `ADMINDN = "cn=root"`
 - `ADMINPW = root`
 - `LDBM_SUFFIX = "o=ibm,c=us"`

- LDBM_DATABASEDIRECTORY = /u/ldap8001/ldb
- SCHEMAPATH = /u/ldap8001/schema
- PROG_SUFFIX = NP
- LDAPUSRID = ldap8001
- LDAPUSRGRP = LDAPGRP
- OUTPUT_DATASET = LDAP8001.CNFOUT
- OUTPUT_DATASET_VOLUME = BH8ST4
- Added jobcard statements for APF_JOB CARD, PRGCTRL_JOB CARD, DB2_JOB CARD, and RACF_JOB CARD
- SLAPD_PROFILE = /u/ldap8001/ds.slapped.profile
- DB2_PROFILE = /u/ldap8001/ds.db2.profile
- RACF_PROFILE = /u/ldap8001/ds.racf.profile

3. Modify ds.slapped.profile with the following changes:

- ARMNAME = LDP8001
- LISTEN = ldap://:4389 (this port must be unique for each ldap running on a single system)
- LOGFILE = /u/ldap8001/logs/gldlog.output

Note: The /logs subdirectory will not be created by IBM Tivoli Directory Server, and must exist prior to starting IBM Tivoli Directory Server. Also verify that the LDAP server's user ID has write access to this entire directory.

4. Modify ds.racf.profile with the following changes:

- LDAPGID = 22 (group ID must be a unique decimal number. Do not use gid 0.)
- LDAPUID = 224 (user ID must be unique decimal number. Do not use uid 0.)

5. Prior to running **dsconfig**, the following variables must be exported on the z/OS UNIX System Services session where **dsconfig** will be run:

- export STEPLIB=SYS1.SIEALNKE:\$STEPLIB
- export PATH=/usr/lpp/ldap/sbin:\$PATH
- export NLSPATH=/usr/lpp/ldap/lib/nls/msg/%L/%N:\$NLSPATH
- export LANG=En_US.IBM-1047

6. Run **dsconfig** with the ds.profile file that was customized:

```
dsconfig -i /u/ldap8001/ds.profile
```

Expect the following messages to be produced:

```
100623 14:20:06.502052 GLD2002I Directory Server configuration utility has started.
```

```
100623 14:20:06.873252 GLD2003I Directory Server configuration utility has ended.
```

Note: You might see the following when running **dsconfig** from an rlogin session:

```
alloc DA('LDAP8001.CNFOUT') RECFM(F,B) LRECL(80) SPACE(6,1) DSNTYPE(PDS)
TRACKS DSORG(PO) BLKSIZE(3200) DIR(10) VOL(BH8ST4)
free DA('LDAP8001.CNFOUT')
IKJ56247I DATA SET LDAP8001.CNFOUT NOT FREED, IS NOT ALLOCATED
RC(12)
```

This error return code of 12 can be ignored. The error is caused when both **dsconfig** and the rlogin environment free the OUTPUT_DATASET. No data is lost.

7. Access the PDS in TSO that was created by **dsconfig**. You should find the following members were created:

- RACF: Creates the `ldap8001` user and `ldapgrp` group and grants authority to RACF-protected resources
 - `PROGxx`: APF-authorizes certain datasets for `ldap`
 - APF: Issues a console command `SET PROG=XX`, points to `progxx` member that was created by `dsconfig`
 - `LDAP8001`: The proc to run the `ldap` started task.
 - `PRGCNTRL`: Sets program control for related libraries
 - `DSCONFIG`: The configuration file for `ldap`
 - `DSENVVAR`: Environment variables that must be used for running `ldap`
8. Copy **PROGxx** to a system `parmlib`.
 9. Submit the job APF, which was created by `dsconfig`, to authorize the datasets listed in `PROGxx`.

Important: The datasets **must** be added to the system APF list to make authorizations persistent across IPLs.

10. Submit the job **RACF** to grant the RACF permissions necessary for the server to run. If using a security product other than RACF, then make the necessary modifications to this job prior to submitting.
11. Copy the LDAP started task JCL (in our example, `LDAP8001`) to a system `proclib`.

9.1.3 Starting and verifying IBM Tivoli Directory Server operation

To start IBM Tivoli Directory Server, issue a console start command for the started task JCL that was created by `dsconfig` and you copied to a system `proclib`. Refer to the comments within the started task proc for parameters that can be specified when starting IBM Tivoli Directory Server. To start IBM Tivoli Directory Server with no parameters, enter `S LDAP8001` on the console, replacing `LDAP8001` with the name of your server in `proclib`.

Verify successful start by watching for the following messages:

```
GLD1004I LDAP server is ready for requests.
GLD1059I Listening for requests on 9.12.4.45 port 4389.
GLD1059I Listening for requests on 9.12.4.46 port 4389.
GLD1059I Listening for requests on 9.12.5.26 port 4389.
GLD1059I Listening for requests on 127.0.0.1 port 4389.
GLD6051I No database changes to commit for LDBM back end named LDBM-0001.
```

To verify IBM Tivoli Directory Server functionality, enter the following command from z/OS UNIX System Services:

```
ldapsearch -h 127.0.0.1 -p 4389 -s base -b "" "objectclass=*"
```

9.2 A basic IBM Tivoli Directory Server server with TDBM

The TDBM back end is based on DB2 and is a highly scalable database implementation. There is no restriction on the type of information that can be stored in the LDBM. DB2 is required to use TDBM.

This section describes the steps needed to implement IBM Tivoli Directory Server using TDBM.

The IBM Tivoli Directory Server installation utility **dsconfig** is used to build a PDS containing the jobs, configuration files, and the **ldap**-started task needed to run IBM Tivoli Directory Server.

9.2.1 Prepare the z/OS system

The following steps need to be done prior to implementing IBM Tivoli Directory Server:

- ▶ UNIX System Services must be operating in full-function mode. There must be sufficient space for storing the schema in the file system.
- ▶ IBM Tivoli Directory Server must be installed using the standard SMP/E process.
- ▶ DB2 v8 or higher is needed to support TDBM as the back end datastore for IBM Tivoli Directory Server. The system on which this work is performed is running DB2 v9.1.

Refer to *z/OS Program Directory* and *z/OS Distributed File Service zSeries File System Administration* for information about installing IBM Tivoli Directory Server and implementing z/OS UNIX System Services.

- ▶ Workload Manager must be implemented to be able to prioritize IBM Tivoli Directory Server with other work running on the system.
- ▶ **dsconfig** assumes that RACF is the security product in use on the system. If another security product is being used, the generated job will need to be modified accordingly.
- ▶ Decide on a naming convention for your IBM Tivoli Directory Server servers. The convention used in our examples is LDAPxxyy, where xx is the LPAR ID and yy is a unique instance for this server.

9.2.2 DB2 setup for IBM Tivoli Directory Server

After DB2 is installed, ask the DB2 system administrator for a user ID that has SYSADM authority. This is needed to submit the JCL associated with setting up IBM Tivoli Directory Server with TDBM. You will also need to get the following information from the system administrator:

- ▶ DB2 subsystem name. For example, D9NG.
- ▶ DB2 server location (or data source). For example, LOC1.

Regardless of whether your DB2 database is local or remote (i.e., on the same z/OS system or a separate z/OS system), you still need to specify the LOCATION. Your database administrator can provide this to you, or you can issue the SPUFI command shown in Figure 9-1. The results of the command are shown in Figure 9-2.

```

DB2 COMMANDS                      SSID: D9NG
===>

Position cursor on the command line you want to execute and press ENTER

Cmd 1 ===> -DISPLAY DDF
Cmd 2 ===>
Cmd 3 ===>
    ...>
Cmd 4 ===>
    ...>
Cmd 5 ===>
    ...>
    ...>
Cmd 6 ===>
    ...>
    ...>
Cmd 7 ===>
    ...>
    ...>
    ...>

PRESS:  ENTER to process   END to save and exit   HELP for more information

```

Figure 9-1 Display DB2 Location

```

DSNL080I  -D9N2 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I  STATUS=STARTD
DSNL082I  LOCATION          LUNAME          GENERICLU
DSNL083I  DB9N              USIBMSC.SCPD9N2  -NONE
DSNL084I  TCPPOPT=37824  SECPOR=0      RESPOR=37826  IPNAME=-NONE
DSNL085I  IPADDR=::9.12.4.47
DSNL086I  SQL      DOMAIN=wtsc81.itso.ibm.com
DSNL086I  RESYNC  DOMAIN=wtsc81.itso.ibm.com
DSNL089I  MEMBER IPADDR=::9.12.4.47
DSNL099I  DSNLTDDF DISPLAY DDF REPORT COMPLETE

```

Figure 9-2 DDF report

Local and remote databases: The DDF component does not need to be started for IBM Tivoli Directory Server if the database is local. If you are using a DB2 database that is on a remote system, the DDF component of DB2 must be configured and started on systems using the DB2 Call Level Interface (CLI). CLI is used by the LDAP server for requesting services from DB2. The DB2 Call Level Interface is the IBM callable SQL interface used by the DB2 family of products, based on the ISO Call Level Interface Draft International Standard specification and the Microsoft Open Database Connectivity specification.

DB2 buffer pools, communication threads and TEMP space might need to be configured for your configuration. Refer to 3.2.5, “Tuning the TDBM back end” on page 37 for more information about performance tuning.

9.2.3 Implementing IBM Tivoli Directory Server with dsconfig

Refer to chapter 5 of *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for more information about each of these steps.

1. Copy the following sample files from `/usr/lpp/ldap/etc` to a new location. You can use `/etc`. However, because we are implementing multiple instances of IBM Tivoli Directory Server on a single system, we chose to create a home directory under `/u` for each instance.

- `ds.profile`
- `ds.slapped.profile`
- `ds.racf.profile`
- `ds.db2.profile` (this file is not changed in this implementation, but `ds.profile` has a link to it)

2. Modify `ds.profile` with the following changes:

- `ADMINDN = "cn=root"`
- `ADMINPW = "root"`
- `TDBM_SUFFIX = "o=ibm,c=us"`
- `PROG_SUFFIX = NP`
- `LDAPUSRID = ldap8102`
- `LDAPUSRGRP = LDAPGRP`
- `OUTPUT_DATASET = LDAP8102.CNFOUT`
- `OUTPUT_DATASET_VOLUME = BH8ST4`
- `DSN_SDSNEXITHLQ = DB9N9`
- `SDSNEXITVOL = BH8DB1`
- `DSN_SDSNLOADHLQ = DB9N9`
- `SDSNLOADVOL = BH8DB1`
- `DSN_SDSNDBRMHLQ = DB9N9`
- `DSN_SSID = D9NG`
- `DB2_VERSION = V9`
- Added jobcard statements for `APF_JOB CARD`, `PRGCTRL_JOB CARD`, `DB2_JOB CARD`, and `RACF_JOB CARD`
- `SLAPD_PROFILE = /u/ldap8102/ds.slapped.profile`
- `DB2_PROFILE = /u/ldap8102/ds.db2.profile`
- `RACF_PROFILE = /u/ldap8102/ds.racf.profile`

3. Modify `ds.slapped.profile` with the following changes:

- `ARMNAME = LDP8001`
- `LISTEN = ldap://:4390` (this port must be unique for each ldap running on a single system)
- `LOGFILE = /u/ldap8102/logs/gldlog.output`

Note: The `/logs` subdirectory will not be created by IBM Tivoli Directory Server, and must exist prior to starting IBM Tivoli Directory Server. Also verify that the LDAP server's user ID has write access to this entire directory.

4. Modify `ds.db2.profile` with the following changes:

- `TDBM_DB2_USERID = GLDSRV` (owner of the tables that will be created)
- `DB2_LOCATION = DB9N`

5. Modify `ds.racf.profile` with the following changes:
 - LDAPGID = 22 (group ID must be a unique decimal number. Do not use gid 0.)
 - LDAPUID = 225 (user ID must be unique decimal number. Do not use uid 0.)
6. Prior to running **dsconfig**, the following variables must be exported on the z/OS UNIX System Services session where **dsconfig** will be run:
 - export STEPLIB=SYS1.SIEALNKE:\$STEPLIB
 - export PATH=/usr/lpp/ldap/sbin:\$PATH
 - export NLSPATH=/usr/lpp/ldap/lib/nls/msg/%L/%N:\$NLSPATH
 - export LANG=En_US.IBM-1047
7. Run **dsconfig** with the `ds.profile` file that was customized:

```
dsconfig -i /u/ldap8102/ds.profile
```

Expect the following output:

```
100629 09:34:47.701052 GLD2002I Directory Server configuration utility has
started.
100629 09:34:49.196760 GLD2003I Directory Server configuration utility has
ended
```

Note: you might see the following when running **dsconfig** from an rlogin session:

```
alloc DA('LDAP8102.CNFOUT') RECFM(F,B) LRECL(80) SPACE(6,1) DSNTYPE(PDS)
TRACKS DSORG(PO) BLKSIZE(3200) DIR(10) VOL(BH8ST4)
free DA('LDAP8102.CNFOUT')
IKJ56247I DATA SET LDAP8102.CNFOUT NOT FREED, IS NOT ALLOCATED
RC(12)
```

This error return code of 12 can be ignored. The error is caused when both **dsconfig** and the rlogin environment free the `OUTPUT_DATASET`. No data is lost

8. Access the PDS in TSO that was created by **dsconfig**. You should find the following members were created:
 - RACF: Creates the `ldap8102` user and `ldapgrp` group, and grants authority to RACF-protected resources
 - PROGxx: APF-authorizes certain datasets for `ldap`
 - APF: Issues a console command `SET PROG=XX` that points to the `progxx` member that was created by **dsconfig**
 - LDAP8001: The proc to run the `ldap`-started task
 - DSCONFIG: Configuration file for `ldap`
 - DSENVVAR: Environment variables that must be used for running `ldap`
 - DBCLI: binds the CLI packages to DB2 and the `DSNACLI` plan
 - PRGCNTRL: Sets program control for related libraries
 - DSNAOINI: DB2 CLI initialization file
 - TDBSPUFI: DB2 DDL needed to create the database, tablespaces, tables, and indexes, and the associated GRANT statements.
9. Copy `PROGxx` to a system `parmlib`.
10. Submit job `APF`, which was created by **dsconfig**, to authorize the datasets listed in `PROGxx`.

Note: The datasets must be added to a the system APF list to make authorizations persistent across IPLs.

11. Submit job RACF to grant the RACF permissions necessary for the server to run. If using a security product other than RACF, then make the necessary modifications to this job prior to submitting.
12. Copy the LDAP started task JCL (in our example, LDAP8102) to a system proclib.
13. Edit member DBCLI. Read the following note prior to submitting this job.

Note: You MUST add a JOBLIB/STEPLIB to run this job, if not already defined in your active linklist configuration, as follows:

```
//JOBLIB DD DISP=SHR,
//          DSN=DB9N9.SDSNLOAD
```

14. Submit DDL using DB2 SPUFI interface as shown in Figure 9-3.

```
SPUFI                                SSID: D9NG
====>

Enter the input data set name:        (Can be sequential or partitioned)
 1 DATA SET NAME ... ==> 'LDAP8102.CNFOUT(TDBSPUFI)'
 2 VOLUME SERIAL ... ==>          (Enter if not cataloged)
 3 DATA SET PASSWORD ==>         (Enter if password protected)

Enter the output data set name:       (Must be a sequential data set)
 4 DATA SET NAME ... ==> 'tdbspufi.output'

Specify processing options:
 5 CHANGE DEFAULTS ==> YES         (Y/N - Display SPUFI defaults panel?)
 6 EDIT INPUT ..... ==> YES        (Y/N - Enter SQL statements?)
 7 EXECUTE ..... ==> YES           (Y/N - Execute SQL statements?)
 8 AUTOCOMMIT ..... ==> YES        (Y/N - Commit after successful run?)
 9 BROWSE OUTPUT ... ==> YES        (Y/N - Browse output data set?)

For remote SQL processing:
10 CONNECT LOCATION ==>

PRESS:  ENTER to process   END to exit           HELP for more information
```

Figure 9-3 DB2 SPUFI DDL submission

All statements should complete with SQLCODE=0. Review and revise the DDL for any that generated a nonzero SQLCODE as shown in Figure 9-4.

```

BROWSE      TDBSPUFI.OUTPUT                               Line 00000082 Col 001 080
Command ==>                                           Scroll ==> PAGE
--*   If necessary, the Database Administrator must manually
--*   update LDAP8102.CNFOUT(TDBSPUFI)
--*   with PRIQTY and SECQTY information for those statements.
--*****

CREATE DATABASE GLDDB STOGROUP SYSDEFLT CCSID EBCDIC;
-----+-----+-----+-----+-----+-----+-----+-----+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+-----+-----+-----+

CREATE TABLESPACE ENTRYTS IN GLDDB
      USING STOGROUP SYSDEFLT
      PRIQTY 14400
      SECQTY 7200
      BUFFERPOOL BPO;
-----+-----+-----+-----+-----+-----+-----+-----+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+-----+-----+-----+

CREATE TABLESPACE LENTRYTS IN GLDDB
      USING STOGROUP SYSDEFLT
      PRIQTY 14400
      SECQTY 7200
      BUFFERPOOL BPO;
-----+-----+-----+-----+-----+-----+-----+-----+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+-----+-----+-----+

```

Figure 9-4 SPUFI output for TDBM

9.2.4 Starting and verifying IBM Tivoli Directory Server operation

To start IBM Tivoli Directory Server, issue a console start command for the started task JCL that was created by **dsconfig** and you copied to a system `proclib`. Refer to the comments within the started task `proc` for parameters that can be specified when starting IBM Tivoli Directory Server. To start IBM Tivoli Directory Server with no parameters, enter **S LDAP8102** on the console, replacing **LDAP8102** with the name of your server in `proclib`.

Test the operation with the following command and watch for the subsequent messages:

```
1dapsearch -h 127.0.0.1 -p 4390 -v 3 -s base -b "" "objectclass=*"
```

Substitute your host for **127.0.0.1** and the correct listening port for **4390**.

```

vendorname=International Business Machines (IBM)
vendorversion=z/OS V1R12
ibmdirectoryversion=z/OS V1R12
subschemasubentry=cn=schema
supportedldapversion=2
supportedldapversion=3
supportedcontrol=1.3.18.0.2.10.20
supportedcontrol=2.16.840.1.113730.3.4.3

```



```
supportedcontrol=2.16.840.1.113730.3.4.2
supportedcontrol=1.3.18.0.2.10.10
supportedcontrol=1.3.18.0.2.10.11
supportedcontrol=1.3.18.0.2.10.15
```

9.3 Set up file-based GDBM to track changes

This section shows how to set up a file-based GDBM to track changes to the server database.

To enable GDBM for a LDBM IBM Tivoli Directory Server instance, perform the following:

1. Stop the LDAP server:
P LDAP8001
2. Open LDAP8001.CNFOUT(DSCONFIG).
3. Uncomment the line database GDBM GLDBGD31/GLDBGD64 as shown in Figure 9-5.

Note: Do not uncomment the line database GDBM GLDBGD31 because this is used for the DB2-based GDBM back end section of the configuration file.

```
EDIT          LDAP8001.CNFOUT(DSCONFIG) - 01.01          Columns 00001 00072
Command ==>>>                                         Scroll ==>>> CSR
003669 #-----
003670 # database dbtype dblibpath
003671 #
003672 # Description:
003673 #   The database option marks the beginning of a new database section.
003674 #
003675 # Example:
003676 #   database GDBM GLDBGD31/GLDBGD64
003677 #
003678 # Notes:
003679 #   All global options must appear before the first database section.
003680 #   An optional name may be specified to identify this back end.
003681 #-----
003682 database GDBM GLDBGD31/GLDBGD64
003683
003684 #-----
```

Figure 9-5 DSCONFIG example of database GDBM

- Specify the database directory where IBM Tivoli Directory Server server will create GDBM / change log related files. If a specified directory is not available, then the IBM Tivoli Directory Server instance will create them. See Figure 9-6.

```

EDIT      LDAP8001.CNFOUT(DSCONFIG) - 01.02          Columns 00001 00072
Command ==>                                         Scroll ==> CSR
003815
003816 #-----
003817 # databaseDirectory <name>
003818 #
003819 # Default Value: /var/ldap/gdbm
003820 #
003821 # Description:
003822 #   The databaseDirectory option specifies the name of the file system
003823 #   directory containing the data files used by this back end, to store
003824 #   the directory data.
003825 #
003826 # Example:
003827 #   databaseDirectory /home/myLdap/gdbmData
003828 #
003829 # Notes:
003830 #   A fully-qualified directory path must be specified. A unique file
003831 #   system directory must be specified for each file-based back end.
003832 #   In addition, when multi-server mode is active, the same directory
003833 #   path must be specified for each instance of this back end within the
003834 #   cross-system group.
003835 #-----
003836 databaseDirectory /u/ldap8001/gdbm
003837
003838 #-----

```

Figure 9-6 DSCONFIG example of databaseDirectory

- Start the LDAP server:

S LDAP8001

- To verify that GDBM is setup correctly, create an ldif file to test logging. Name it **addpassword.ldif**.

```

dn : cn=Bob Garcia, ou=Poughkeepsie, o=ibm,c=us
changetype : modify
add : userpassword
userpassword : passw0rd

```

Use the **ldapmodify** command to load the addpassword.ldif file:

```

ldapmodify -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -p 4389 -f
addpassword.ldif 2>&1 | tee addpassword.out

```

Look for the following:

```

modifying entry cn=Bob Garcia, ou=Poughkeepsie, o=ibm,c=us

```

- Review the changelog record. Use **ldapsearch** command with cn=changelog as a base.

```

ldapsearch -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -p 4389 -s sub -b
"cn=changelog" objectclass=*

```

Look for the following:

```

changeNumber=1,cn=changelog
objectclass=top

```

```

objectclass=changeLogEntry
objectclass=ibm-changeLog
changenumber=1
changetype=modify
targetdn=cn=Bob Garcia, ou=Poughkeepsie, o=ibm,c=us
changes=add: userpassword
userpassword: *ComeAndGetIt*
-
ibm-changeinitiatorsname=cn=LDAP Admin, o=ibm, c=us
changetime=20100624184719.357902Z

cn=changelog
objectclass=top
objectclass=container
cn=changelog

```

9.4 Set up DB2-based GDBM to track changes

This section shows how to set up a DB2-based GDBM to track changes to the server database. When using DB2 to store its entries, the GDBM database is identical to a TDBM database and is created in the same way using the same SPUFI script. A DB2-based GDBM back end cannot share a database with a TDBM back end. Like TDBM, a DB2-based GDBM back end cannot run in 64-bit mode.

To enable GDBM for a TDBM IBM Tivoli Directory Server instance, do the following:

1. Stop the LDAP server:


```
P LDAP8102
```
2. Open LDAP8102.CNFOUT(DSCONFIG).
3. Uncomment the line `database GDBM GLDBGD31` and specify a database name as shown in Figure 9-7.

```

EDIT      LDAP8102.CNFOUT(DSCONFIG) - 01.05          Columns 00001 00072
Command ==>                                         Scroll ==> CSR
002006 #-----
002007 # database dbtype dblibpath
002008 #
002009 # Description:
002010 #   The database option marks the beginning of a new database section.
002011 #
002012 # Example:
002013 #   database GDBM GLDBGD31
002014 #
002015 # Notes:
002016 #   All global options must appear before the first database section.
002017 #   An optional name may be specified to identify this back end.
002018 #-----
002019 database GDBM GLDBGD31 GDBMDB

```

Figure 9-7 DSCONFIG example of database

Note: Do not uncomment the line database GDBM GLDBGD31/GLDBGD64 because this is used for the file-based GDBM back end section of the configuration file.

4. Uncomment the line `dbuserid` and specify a unique name that must be different than the `dbuserid` value for the TDBM back end. An example is shown in Figure 9-8.

```
EDIT      LDAP8102.CNFOUT(DSCONFIG) - 01.05          Columns 00001 00072
Command ==>                                         Scroll ==> CSR
002153 #-----
002154 # dbuserid <userid>
002155 #
002156 # Description:
002157 #   The dbuserid option specifies the z/OS user ID that will be the
002158 #   owner of the DB2 tables. This option indicates that this GDBM
002159 #   back end is DB2-based and not file-based.
002160 #
002161 # Example:
002162 #   dbuserid LDAPSRV
002163 #
002164 # Notes:
002165 #   This option is required when using the DB2-based GDBM back end.
002166 #   this value must be unique within this configuration file.
002167 #   Multiple back ends on an LDAP server cannot share a database.
002168 #-----
002169 dbuserid GLDSRVG
```

Figure 9-8 DSCONFIG example of `dbuserid`

5. Because the database layout for GDBM is identical to the layout for TDBM, use the same DDL that was created for TDBM.
6. Update all occurrences of the database owner to match the `dbuserid` that was specified in DSCONFIG.
7. Update all occurrences of the database name to match the name that was specified for database GDBM GLDBGD31 in DSCONFIG

8. Specify multiserver on in the GDBM-specific section of DSCONFIG as shown in Figure 9-9.

```

EDIT          LDAP8102.CNFOUT(DSCONFIG) - 01.07          Columns 00001 00072
Command ==>          Scroll ==> CSR
002292 #-----
002293 # multiserver <on | off>
002294 #
002295 # Default Value: off
002296 #
002297 # Description:
002298 #   The multiserver option specifies the operating mode for this
002299 #   back end.
002300 #
002301 # Example:
002302 #   multiserver on
002303 #
002304 # Notes:
002305 #   You can configure a back end to operate in single-server mode while
002306 #   another back end operates in multi-server mode except when GDBM or
002307 #   CDBM is configured. When CDBM or GDBM is configured, all TDBM,
002308 #   LDBM, GDBM, and CDBM back ends must be configured to use the same
002309 #   operating mode.
002310 #-----
002311 multiserver on

```

Figure 9-9 Specifying multiserver on for gdbm

9. Submit the DDL using DB2 SPUFI as shown in Figure 9-10.

```

SPUFI          SSID: D9NG
==>

Enter the input data set name:      (Can be sequential or partitioned)
1  DATA SET NAME ... ==> 'LDAP8102.CNFOUT(GDBSPUFI)'
2  VOLUME SERIAL ... ==>          (Enter if not cataloged)
3  DATA SET PASSWORD ==>         (Enter if password protected)

Enter the output data set name:     (Must be a sequential data set)
4  DATA SET NAME ... ==> 'TDBSPUFI.OUTPUT'

Specify processing options:
5  CHANGE DEFAULTS ==> YES        (Y/N - Display SPUFI defaults panel?)
6  EDIT INPUT ..... ==> YES      (Y/N - Enter SQL statements?)
7  EXECUTE ..... ==> YES         (Y/N - Execute SQL statements?)
8  AUTOCOMMIT ..... ==> YES      (Y/N - Commit after successful run?)
9  BROWSE OUTPUT ... ==> YES     (Y/N - Browse output data set?)

For remote SQL processing:
10 CONNECT LOCATION ==>

PRESS:  ENTER to process   END to exit           HELP for more information

```

Figure 9-10 SPUFI panel for GDBM DDL

All SQL codes should equal zero. If any do not, review the DDL to determine the reason as shown in Figure 9-11.

```
BROWSE      TDBSPUFI.OUTPUT                      Line 00000084 Col 001 080
Command ==>                                     Scroll ==> PAGE
--*   with PRIQTY and SECQTY information for those statements.
--*****

CREATE DATABASE GDBMDB STOGROUP SYSDEFLT CCSID EBCDIC;
-----+-----+-----+-----+-----+-----+-----+-----+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+-----+-----+-----+

CREATE TABLESPACE ENTRYTS IN GDBMDB
      USING STOGROUP SYSDEFLT
      PRIQTY 14400
      SECQTY 7200
      BUFFERPOOL BPO;
-----+-----+-----+-----+-----+-----+-----+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+-----+-----+-----+

CREATE TABLESPACE LENTRYTS IN GDBMDB
      USING STOGROUP  SYSDEFLT
      PRIQTY 14400
      SECQTY 7200
      BUFFERPOOL BPO;
-----+-----+-----+-----+-----+-----+-----+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+-----+-----+-----+

CREATE TABLESPACE LATTRTS IN GDBMDB
      USING STOGROUP  SYSDEFLT
```

Figure 9-11 SPUFI output for GDBM

10. Stop all instances of IBM Tivoli Directory Server that are sharing the same TDBM.
11. Make the same DSCONFIG updates for all instances of IBM Tivoli Directory Server that are sharing the same TDBM, if you are using a different DSCONFIG file for each instance.
12. Restart all instances of IBM Tivoli Directory Server.

9.5 A basic IBM Tivoli Directory Server server with SDBM

RACF (Resource Access Control Facility) is a security system that provides access control and auditing functionality for the z/OS and z/VM® operating systems.

RACF provides:

- ▶ Identification and verification of a user using user ID and password check (authentication)
- ▶ Protection of resources by maintenance of access rights (authorization)
- ▶ Logging of accesses to protected resources (auditing)

The z/OS IBM Tivoli Directory Server server can provide remote LDAP access to the user, group, connection, and general resource profile information stored in RACF. It also supports

setting RACF options that affect classes. Using SDBM, the RACF database back end of the LDAP provides these features:

- ▶ Authentication of RACF users.
- ▶ Add, modify, and delete RACF users, groups, and general resources. Note that dataset resources are not supported.
- ▶ Add, modify, and delete user connections to groups.
- ▶ Add and remove users and groups in general resource access lists.
- ▶ Modify SETROPTS options that affect classes (for example, RACLIST).
- ▶ Retrieve RACF information for users, groups, connections, general resources, and class options.
- ▶ Retrieve RACF user password and password phrase envelopes.

z/OS IBM Tivoli Directory Server automatically provides the attributes and objectclasses used by SDBM. However, attributes need to be added to schema to manage RACF custom fields with the LDAP server.

To configure your LDAP server to run with the SDBM back end of the LDAP server:

1. Stop the LDAP Server:

```
P LDAP8001
```

2. Edit the configuration file and enable SDBM as shown in Figure 9-12.

```
EDIT      LDAP8001.CNFOUT(DSCONFIG) - 01.07          Columns 00001 00072
Command ==>                                         Scroll ==> CSR
001175
001176 #-----
001177 # database dbtype dblibpath
001178 #
001179 # Description:
001180 #   The database option marks the beginning of a new database section.
001181 #
001182 # Example:
001183 #   database SDBM GLDBSD31/GLDBSD64
001184 #
001185 # Notes:
001186 #   All global options must appear before the first database section.
001187 #   An optional name may be specified to identify this back end.
001188 #-----
001189 database SDBM GLDBSD31/GLDBSD64
```

Figure 9-12 Adding SDBM to the configuration

3. Specify the SDBM suffix as shown in Figure 9-13 on page 192.

Note: SDBM suffix must be a unique suffix and it should not overlap with existing suffixes.

```

EDIT          LDAP8001.CNFOUT(DSCONFIG) - 01.08          Columns 00001 00072
Command ==>          Scroll ==> CSR
001202
001203 #-----
001204 # suffix <dn-suffix>
001205 #
001206 # Description:
001207 #   The suffix option specifies the root of a subtree in the namespace
001208 #   managed by this server within this back end.
001209 #
001210 # Example:
001211 #   suffix "sysplex=sysplex1"
001212 #
001213 # Notes:
001214 #   This option is required when using the SDBM back end.
001215 #-----
001216 suffix "cn=RACF,o=ibm,c=in"

```

Figure 9-13 Adding the SDBM suffix

4. Save the configuration file.
5. Start the server:

```
S LDAP8001
```

6. Use the following command to retrieve SDBM back end data:

```
ldapsearch -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -p 4389 -s base -b
"cn=RACF,o=ibm,c=in" objectclass=*
```

9.6 Loading the IBM-supplied schema

z/OS IBM Tivoli Directory Server stores the schema as an entry in the database, and the distinguished name of the schema entry is `cn=schema`. Use `ldapsearch` with `cn=schema` base to list the schema entry in the running IBM Tivoli Directory Server instance:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s base -b "cn=schema"
objectclass=*
```

Replace `ldaphost` with your host name, `port` with the correct the port number, and `adminDN` and `passwd` with the administrator distinguished name and password.

For example:

```
ldapsearch -D cn=root -w root -p 4389 -s base -b "cn=schema" objectclass=*
```

The value of the `schemaPath` option in the IBM Tivoli Directory Server for z/OS configuration file defines the location where IBM Tivoli Directory Server stores the schema entry. The z/OS IBM Tivoli Directory Server instance owner needs to have read and write permission to the schema location. The default value is `/var/ldap/schema`. If multiple instances of IBM Tivoli Directory Server are running on one system in single-server mode, update the `schemaPath` configuration option to specify a separate directory location for each server that is running. If running multiple servers on a system in multi-server mode, the `schemaPath` configuration option must be the same in all configuration files and the schema directory location must reside in a shared file system.

On first start-up IBM Tivoli Directory Server creates an initial default schema that is sufficient for usage of the GDBM, CDBM, and SDBM (w/o custom fields), but needs to be updated for usage of LDBM, TDBM, SDBM with RACF custom fields, and CDBM with user-defined entries.

z/OS IBM Tivoli Directory Server is shipped with the predefined schema files `schema.IBM.ldif` and `schema.user.ldif`. Use the **ldapmodify** command to load the schema files to the running IBM Tivoli Directory Server instance. The commands to load the `schema.user.ldif` and `schema.IBM.ldif` schema files are:

```
ldapmodify -h ldaphost -p ldapport -D adminDN -w passwd -f schemaFile
```

Replace *ldaphost* with your host name, *ldapport* with the correct portnumber, and *adminDN* and *passwd* with the administrator distinguished name and password.

For example:

```
ldapmodify -D cn=root -w root -p 4389 -f /usr/lpp/ldap/etc/schema.user.ldif  
modifying entry cn=schema
```

```
ldapmodify -D cn=root -w root -p 4389 -f /usr/lpp/ldap/etc/schema.IBM.ldif  
modifying entry cn=schema
```

9.7 Loading the IBM-supplied sample.ldif file

The **ldapadd** command is used to populate IBM Tivoli Directory Server with data. A sample file is included to populate and test IBM Tivoli Directory Server. The file is located in `/usr/lpp/ldap/examples/sample_server/sample.ldif`.

Use the **ldapadd** command as follows to load the sample data from `sample.ldif`:

```
ldapadd -p 4389 -D cn=root -w root -f sample.ldif
```

Expect to see output similar to the following:

```
adding new entry o=ibm,c=us  
adding new entry ou=Poughkeepsie, o=ibm,c=us  
adding new entry ou=In Flight Systems, ou=Poughkeepsie, o=ibm,c=us  
adding new entry ou=Home Entertainment, ou=Poughkeepsie, o=ibm,c=us  
adding new entry ou=Groups, o=ibm,c=us  
adding new entry cn=Bowling team, ou=Groups, o=ibm,c=us  
adding new entry ou=Widget Division, ou=Poughkeepsie, o=ibm,c=us  
adding new entry cn=Mary Burnnet, ou=Widget Division, ou=Poughkeepsie, o=ibm,c=us  
adding new entry cn=David Campbell, ou=Widget Division, ou=Poughkeepsie,  
o=ibm,c=us  
adding new entry cn=James Campbell, ou=Widget Division, ou=Poughkeepsie,  
o=ibm,c=us
```

If your ldif file has a large number of users and group and the data is to be loaded into a TDBM back end, use the **ldif2ds** command to load the data. **ldif2ds** cannot be used to load entries into a GDBM, LDBM, CDBM, or SDBM directory. The IBM Tivoli Directory Server instance should be stopped while running **ldif2ds**.

The **ldif2ds** command will not replicate data even if a replica is configured.

9.8 Securing the IBM Tivoli Directory Server administration ID

The **adminPW** parameter in the instance configuration file defines the password for the server administrator, who has unrestricted access to all entries in the directory irrespective of the replication configuration and ACLs. z/OS IBM Tivoli Directory Server stores the value of the **adminPW** parameter in cleartext, which can be a security risk. Therefore, do not specify the admin password in the instance configuration file after the Directory Information Tree (DIT) is configured.

Another and more secure way is for the user to define **adminDN** as an entry in the back end under the predefined suffix entry. The **userPassword** attribute is used to hold the password for the administrator in this case. The encryption method given in the value of the **pwEncryption** parameter is used to encrypt the **userPassword** attribute.

Use the following steps to secure the administrator distinguished name for the running IBM Tivoli Directory Server instance.

1. Create an `ldif` file with the following contents, and name it `admin.ldif`:

```
dn: cn=LDAP Admin, o=ibm, c=us
objectclass: person
cn: LDAP Admin
description: Administrator DN for the server
sn: Administrator
userpassword: sec001ret
```

Important: Do not use this example without changing the password value, and the actual distinguished name.

2. Load the `admin.ldif` file. Use the **ldapadd** command to load the file:

```
ldapadd -h ldaphost -p port -D adminDN -w passwd -f admin.ldif
```

For example:

```
ldapadd -p 4389 -D cn=root -w root -f admin.ldif
```

3. Stop the IBM Tivoli Directory Server instance from the MVS console:

```
P LDAP8001
```

Watch for the following messages to confirm the server is stopped:

```
GLD1006I LDAP server stop command received.
```

```
GLD1007I LDAP server is stopping.
```

```
GLD6033I Committing changes to database for LDBM back end named LDBM-0001.
```

```
GLD6034I Completed committing changes to database for LDBM back end named LDBM-0001.
```

- Update the IBM Tivoli Directory Server configuration file that is used at **ldap** startup. Figure 9-14 uses the PDS file LDAP8001.CNFOUT(DSCONFIG). Specify the value for **AdminDN** that you specified for **dn:** in the `admin.ldif` file. There is no need to edit the **adminPW** parameter.

```

EDIT          LDAP8001.CNFOUT(DSCONFIG) - 01.00          Columns 00001 00072
Command ==>
000070 #   adminDN "cn=LDAP Administrator"
000071 #
000072 # Recommendation:
000073 #   After the schema is loaded it is recommended, though not necessary,
000074 #   that the DN be loaded into the directory with the same suffix as
000075 #   one of the suffix option values below.
000076 #
000077 #   Once the entry is loaded in the directory for this distinguished
000078 #   name the adminDN should be changed to reflect the new DN. The
000079 #   entry is used when evaluating an LDAP bind operation for the
000080 #   adminDN.
000081 #
000082 # Example:
000083 #   adminDN "cn=Admin, o=Your Company"
000084 #
000085 # Notes:
000086 #   This configuration option must be specified.
000087 #-----
000088 adminDN "cn=LDAP Admin, o=ibm, c=us"
000089
000090 #-----
000091 # adminPW <password>
000092 #
000093 # Description:
000094 #   The adminPW option specifies the password for the administrator
000095 #   defined by the adminDN option.

```

Figure 9-14 Sample DSCONFIG file showing update for adminDN

- Start the IBM Tivoli Directory Server instance from the MVS console:

```
S LDAP8001
```

You should see the following messages:

```

GLD1004I LDAP server is ready for requests.
GLD1059I Listening for requests on 9.12.4.45 port 4389.
GLD1059I Listening for requests on 9.12.4.46 port 4389.
GLD1059I Listening for requests on 9.12.5.26 port 4389.
GLD1059I Listening for requests on 127.0.0.1 port 4389.
GLD6051I No database changes to commit for LDBM back end named LDBM-0001.

```

- Test the new admin ID with the new password:

```
ldapsearch -p 4389 -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -s base -b
o=ibm,c=us objectclass=*
```

Watch for the following messages:

```

o=ibm,c=us
objectclass=top
objectclass=organization
o=ibm

```

7. Use the `ldapsearch` command to crosscheck the bind with old password, it should return an error:

```
ldapsearch -p 4389 -D "cn=LDAP Admin, o=ibm, c=us" -w root -s base -b  
o=ibm,c=us objectclass=*
```

Watch for the following:

```
ldap_sasl_bind: Credentials are not valid  
ldap_sasl_bind: additional info: R004062 Credentials are not valid  
(process_simple_bind)
```

9.9 Using CRAM-MD5 and DIGEST-MD5 binds

The Digest and Challenge response authentication (CRAM) message digest algorithm 5 are simple authentication and security layer (SASL) authentication mechanisms. When a client uses DIGEST-MD5 or CRAM-MD5, the password in the bind request is not transmitted in clear text and the protocol prevents replay attacks. Both the CRAM-MD5 and DIGEST-MD5 mechanisms are multi-stage binds where the server sends the client a challenge and then the client sends a challenge response back to the server to complete the authentication. The client challenge response contains a hash of the password entered by the user, the username, and other pieces of data encoded to the specifications of either the CRAM-MD5 or DIGEST-MD5 RFCs.

The CRAM-MD5 and DIGEST-MD5 bind mechanisms on the z/OS LDAP server do not require any additional products to be installed or configured, and SASL bind mechanisms are more secure than performing simple binds because the credentials are not passed in clear text.

This section shows how to enable MD5 security checking, using both CRAM-MD5 and DIGEST-MD5. Refer to chapter 20 of *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for more information about this topic.

To enable MD5 security, perform the following steps:

1. Stop the IBM Tivoli Directory Server instance:

```
P LDAP8001
```

2. Open LDAP8001.CNFOUT(DSCONFIG).

- Uncomment the **digestRealm** parameter, and specify the host name of the z/OS on which the IBM Tivoli Directory Server instance is running (Figure 9-15).

```

EDIT          LDAP8001.CNFOUT(DSCONFIG) - 01.02          Columns 00001 00072
Command ==>          Scroll ==> CSR
001021 #-----
001022 # digestRealm <hostname>
001023 #
001024 # Default Value: primary host name obtained from DNS
001025 #
001026 # Description:
001027 #   The digestRealm option specifies a realm name to be used when doing
001028 #   DIGEST-MD5 or CRAM-MD5 SASL authentication binds.
001029 #
001030 # Example:
001031 #   digestRealm host.server.com
001032 #
001033 # Notes:
001034 #   This value is sent to the client to help hash the password while
001035 #   doing a DIGEST-MD5 or CRAM-MD5 SASL bind.
001036 #-----
001037 digestRealm WTSC80.ITS0.IBM.COM

```

Figure 9-15 DSCONFIG digestRealm example

- Save the LDAP8001.CNFOUT(DSCONFIG) file.
- Start the server:


```
S LDAP8001
```
- Specify the username in the bind mechanism using **-U** option of the **ldapsearch** client utility. The username that is specified must map to one of the uid attribute values in one of the TDBM, LDBM, or CDBM entries.

```
ldapsearch -U Nilesh -w secret -p 4389 -s base -b "o=ibm,c=us" objectclass=*
```

You should see the following messages:

```
o=ibm,c=us
objectclass=top
objectclass=organization
o=ibm
```

Note: The uid attribute values specified on the entries to be used for CRAM-MD5 or DIGEST-MD5 authentication must be unique across every TDBM, LDBM, and CDBM back end that is configured on the LDAP server. Authentication fails if more than one entry has the same uid attribute value.

9.10 Enabling SSL authentication

In this section we show how to enable SSL authentication using a self-signed certificate. The process would need to be modified slightly if using an external certificate provider. The other options that are supported by IBM Tivoli Directory Server, in addition to a RACF key ring, are a key database and PKCS #11 token. To enable SSL authentication:

1. The user ID under which the LDAP server runs must be authorized by RACF to use RACF key rings. To authorize the LDAP server, use the following RACF commands:

```
RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(LDAP8001) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(LDAP8001) ACCESS(CONTROL)
SETROPTS RACLIST(FACILITY) REFRESH
```

2. Create a RACF key ring for the IBM Tivoli Directory Server instance:

```
RACDCERT ID(LDAP8001) ADDRING(LDAP8001.KEYRING)
```

Important: The keyring name is case-sensitive. The same case used in the RACDCERT MUST be used when updating the configuration file.

```
RACDCERT ID(LDAP8001) GENCERT SUBJECTSDN(CN('LDAP8001') O('IBM') C('US'))
WITHLABEL('LDAPSSLCERT') KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)
NOTAFTER(2020-12-31)
```

```
SETROPTS RACLIST(DIGTCERT) REFRESH
```

```
RACDCERT ID(LDAP8001) CONNECT(LABEL('LDAPSSLCERT') RING(LDAP8001.KEYRING)
USAGE(PERSONAL))
```

```
SETROPTS RACLIST(FACILITY) REFRESH
```

```
RACDCERT ID(LDAP8001) EXPORT(LABEL('LDAPSSLCERT')) DSN('LDAP8001.LDAP.CERT')
FORMAT(CERTB64)
```

3. Stop the LDAP server if it is running. In this example, the instance name is LDAP8001.
P LDAP8001
4. Open the configuration PDS file, in our example LDAP8001.CNFOUT(DSCONFIG).

5. Add a port for secure communication in the IBM Tivoli Directory Server configuration file as shown in Figure 9-16.

```

EDIT          LDAP8001.CNFOUT(DSCONFIG) - 01.03          Columns 00001 00072
Command ==>          Scroll ==> CSR
000368 #
000369 # Listen for client requests from IPv6 address 1080::8:800:200C:417A
000370 # on the default non-secure port 389:
000371 #   listen ldap://[1080::8:800:200C:417A]
000372 #
000373 # Listen for client requests from IPv6 address 1080::8:800:200C:417B
000374 # on non-secure port 489:
000375 #   listen ldap://[1080::8:800:200C:417B]:489
000376 #
000377 # Listen for client requests from IPv6 address 1080::8:800:200C:417C
000378 # on secure port 436:
000379 #   listen ldaps://[1080::8:800:200C:417C]:436
000380 #
000381 # Notes:
000382 # Only one LDAP server on each system can listen for requests using
000383 # the Program Call interface. The listen option overrides any values
000384 # specified by the deprecated options; security, port and securePort.
000385 #-----
000386 listen ldap://:4389
000387 listen ldaps://:4636
000388
000389 #-----

```

Figure 9-16 Sample DSCONFIG showing update for adding a port for ldaps

6. Add the sslCertificate label to the IBM Tivoli Directory Server configuration file as shown in Figure 9-17.

```

EDIT          LDAP8001.CNFOUT(DSCONFIG) - 01.04          Columns 00001 00072
Command ==>          Scroll ==> CSR
000906
000907 #-----
000908 # sslCertificate <certificate-label | none>
000909 #
000910 # Default Value: none
000911 #
000912 # Description:
000913 # The sslCertificate option specifies the label of the certificate
000914 # that is used for LDAP server authentication.
000915 #
000916 # Example:
000917 #   sslCertificate cert1
000918 #
000919 # Notes:
000920 # If the value is none, the default certificate is used for server
000921 # authentication.
000922 #-----
000923 sslCertificate LDAPSSLCERT
000924
000925 #-----

```

Figure 9-17 Adding sslCertificate

7. Add `sslKeyRingFile` to the configuration file as shown in Figure 9-18.

```

EDIT          LDAP8001.CNFOUT(DSCONFIG) - 01.04          Columns 00001 00072
Command ==>                                         Scroll ==> CSR
000938
000939 #-----
000940 # sslKeyRingFile <filename | keyring | token>
000941 #
000942 # Description:
000943 #   The sslKeyRingFile option specifies the path and file name of the
000944 #   SSL/TLS key database file, RACF key ring name, or PKCS #11 token
000945 #   name for the LDAP server.  SSL/TLS connections are only
000946 #   available when this option is specified.
000947 #
000948 # Examples:
000949 #   sslKeyRingFile /etc/ldap/key.kdb
000950 #   sslKeyRingFile LDAPRING
000951 #   sslKeyRingFile *TOKEN*/MYTOKEN
000952 #
000953 # Notes:
000954 #   The sslKeyRingFilePW and sslKeyRingPWStashFile configuration
000955 #   options must not be specified when a RACF key ring name or
000956 #   PKCS #11 token name is specified for this option.
000957 #-----
000958 sslKeyRingFile LDAP8001.KEYRING
000959

```

Figure 9-18 Adding `sslKeyRingFile`

8. Save the configuration file, then start the server:

```
S LDAP8001
```

You should see output similar to the following:

```

S LDAP8001
$HASP100 LDAP8001 ON STCINRDR IEF695I START LDAP8001 WITH JOBNAME LDAP8001 IS
ASSIGNED TO USER LDAP8001, GROUP LDAPGRP
$HASP373 LDAP8001 STARTED
BPXM023I (LDAP8001) GLD1004I LDAP server is ready for requests.
GLD1005I LDAP server start command processed.
BPXM023I (LDAP8001) GLD1059I Listening for requests on 9.12.4.45 port 4389.
BPXM023I (LDAP8001) GLD1211I Listening for requests on 9.12.4.45 secure port
4636.
BPXM023I (LDAP8001) GLD1059I Listening for requests on 9.12.4.46 port 4389.
BPXM023I (LDAP8001) GLD1211I Listening for requests on 9.12.4.46 secure port
4636.
BPXM023I (LDAP8001) GLD1059I Listening for requests on 9.12.5.26 port 4389.
BPXM023I (LDAP8001) GLD1211I Listening for requests on 9.12.5.26 secure port
4636.
BPXM023I (LDAP8001) GLD1059I Listening for requests on 127.0.0.1 port 4389.
BPXM023I (LDAP8001) GLD1211I Listening for requests on 127.0.0.1 secure port
4636.

```

9. Copy dataset `LDAP8001.LDAP.CERT` to an `LDAPSSLCERT` file in the UNIX file system tree.

10. Using **gskkyman** command to perform the following steps:
- Create a new database (Figure 9-19 and Figure 9-20).

```
NPATEL:/u/npatel/sslkeys: >gskkyman

Database Menu

    1 - Create new database
    2 - Open database
    3 - Change database password
    4 - Change database record length
    5 - Delete database
    6 - Create key parameter file
    7 - Display certificate file (Binary or Base64 ASN.1 DER)

   11 - Create new token
   12 - Delete token
   13 - Manage token
   14 - Manage token from list of tokens

    0 - Exit program

Enter option number: 1
```

Figure 9-19 Create new database

```
Enter key database name (press ENTER to return to menu): ldapcltcert.kdb
Enter database password (press ENTER to return to menu):
Re-enter database password:
Enter password expiration in days (press ENTER for no expiration):
Enter database record length (press ENTER to use 5000):

Enter 1 for FIPS mode database or 0 to continue: 0

Key database /u/npatel/sslkeys/ldapcltcert.kdb created.

Press ENTER to continue.
```

Figure 9-20 Database parameters

- b. Create a self-signed certificate (Figure 9-21, Figure 9-22, Figure 9-23, and Figure 9-24 on page 203).

```
Key Management Menu

    Database: /u/npatel/sslkeys/ldapcltcert.kdb
    Expiration: None

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to previous menu): 6
```

Figure 9-21 Select Create a self-signed certificate

```
Certificate Type

1 - CA certificate with 1024-bit RSA key
2 - CA certificate with 2048-bit RSA key
3 - CA certificate with 4096-bit RSA key
4 - CA certificate with 1024-bit DSA key
5 - User or server certificate with 1024-bit RSA key
6 - User or server certificate with 2048-bit RSA key
7 - User or server certificate with 4096-bit RSA key
8 - User or server certificate with 1024-bit DSA key

Select certificate type (press ENTER to return to menu): 1
```

Figure 9-22 Select certificate type

```
Signature Digest Type

1 - SHA-1
2 - SHA-224
3 - SHA-256
4 - SHA-384
5 - SHA-512

Select digest type (default SHA-1):
```

Figure 9-23 Select Signature Digest type

```

Enter label (press ENTER to return to menu): ldapclt
Enter subject name for certificate
  Common name (required): ldapclt
  Organizational unit (optional): poughkeepsie
  Organization (required): ibm
  City/Locality (optional):
  State/Province (optional):
  Country/Region (2 characters - required): us
Enter number of days certificate will be valid (default 365):

Enter 1 to specify subject alternate names or 0 to continue: 0

Please wait .....

Certificate created.
Press ENTER to continue.

```

Figure 9-24 Enter certificate information

- c. Import an LDAPSSLCERTcertificate (Figure 9-25, Figure 9-26, Figure 9-27 on page 204, and Figure 9-28 on page 204).

```

Key Management Menu

      Database: /u/npatel/sslkeys/ldapcltcert.kdb
      Expiration: None

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to previous menu): 7

```

Figure 9-25 Select import a certificate

```

Enter import file name (press ENTER to return to menu): /u/npatel/cert
Enter label (press ENTER to return to menu): LDAPSSLCERT

Certificate imported.

Press ENTER to continue.

```

Figure 9-26 Import IBM Tivoli Directory Server server authentication certificate

```
Key Management Menu

    Database: /u/npatel/sslkeys/ldapcltcert.kdb
    Expiration: None

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to previous menu):
```

Figure 9-27 Select ENTER to return to previous menu

```
Database Menu

1 - Create new database
2 - Open database
3 - Change database password
4 - Change database record length
5 - Delete database
6 - Create key parameter file
7 - Display certificate file (Binary or Base64 ASN.1 DER)

11 - Create new token
12 - Delete token
13 - Manage token
14 - Manage token from list of tokens

0 - Exit program

Enter option number: 0
NPATEL:/u/npatel/sslkeys: >
```

Figure 9-28 Exit gskkyman

11. Use the **ldapsearch** command to test the server auth SSL configuration:

```
ldapsearch -h ldaphost -p sslport -Z -K keyfile -P keypasswd -D adminDN -w  
passwd -s base -b o=ibm,c=us objectclass=*
```

Example:

```
ldapsearch -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -p 4636 -Z -K  
"/u/npatel/sslkeys/ldapcltcert.kdb" -P "sec001ret" -s base -b "o=ibm,c=us"  
objectclass=*
```

You should see messages similar to the following:

```
o=ibm,c=us
objectclass=top
objectclass=organization
o=ibm
```

12. After server auth SSL completes successfully, use the **gskkyman** command and follow these steps:

a. Open the `ldap1tcert.kdb` database (Figure 9-29 and Figure 9-30).

```
NPATEL:/u/npatel/sslkeys: >gskkyman

      Database Menu

1 - Create new database
2 - Open database
3 - Change database password
4 - Change database record length
5 - Delete database
6 - Create key parameter file
7 - Display certificate file (Binary or Base64 ASN.1 DER)

11 - Create new token
12 - Delete token
13 - Manage token
14 - Manage token from list of tokens

0 - Exit program

Enter option number: 2
```

Figure 9-29 Start gskkyman and select open the database

```
Enter key database name (press ENTER to return to menu): ldap1tcert.kdb
Enter database password (press ENTER to return to menu):
```

Figure 9-30 Enter database information

- b. Create a new certificate (Figure 9-31, Figure 9-32, Figure 9-33 on page 207, and Figure 9-34 on page 207).

```
Key Management Menu

    Database: /u/npatel/sslkeys/ldapcltcert.kdb
    Expiration: None

    1 - Manage keys and certificates
    2 - Manage certificates
    3 - Manage certificate requests
    4 - Create new certificate request
    5 - Receive requested certificate or a renewal certificate
    6 - Create a self-signed certificate
    7 - Import a certificate
    8 - Import a certificate and a private key
    9 - Show the default key
    10 - Store database password
    11 - Show database record length

    0 - Exit program

Enter option number (press ENTER to return to previous menu): 4
```

Figure 9-31 Select Create a new certificate request

```
Certificate Type

    1 - Certificate with 1024-bit RSA key
    2 - Certificate with 2048-bit RSA key
    3 - Certificate with 4096-bit RSA key
    4 - Certificate with 1024-bit DSA key

Enter certificate type (press ENTER to return to menu): 1
Enter request file name (press ENTER to return to menu): ldapcltcert
Enter label (press ENTER to return to menu): ldapcltcert
Enter subject name for certificate
    Common name (required): Nilesch
    Organizational unit (optional): Poughkeepsie
    Organization (required): ibm
    City/Locality (optional):
    State/Province (optional):
    Country/Region (2 characters - required): us

Enter 1 to specify subject alternate names or 0 to continue: 0

Please wait .....

Certificate request created.

Press ENTER to continue.
```

Figure 9-32 Enter certificate information

```
Key Management Menu

Database: /u/npatel/sslkeys/ldapcltcert.kdb
Expiration: None

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to previous menu): 2
```

Figure 9-33 Select Manage certificate requests

```
Key Management Menu

Database: /u/npatel/sslkeys/ldapcltcert.kdb
Expiration: None

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to previous menu): 1
```

Figure 9-34 Return to Key Management menu and select Manage keys and certificates

c. Export certificate to a clientcert.der file (Figure 9-35, Figure 9-36, and Figure 9-37).

```
Key and Certificate List

      Database: /u/npatel/sslkeys/ldapcltcert.kdb

      1 - ldapclt

      0 - Return to selection menu

Enter label number (ENTER to return to selection menu, p for previous list): 1
```

Figure 9-35 Select the certificate to export

```
Key and Certificate Menu

      Label: ldapclt

      1 - Show certificate information
      2 - Show key information
      3 - Set key as default
      4 - Set certificate trust status
      5 - Copy certificate and key to another database/token
      6 - Export certificate to a file
      7 - Export certificate and key to a file
      8 - Delete certificate and key
      9 - Change label
     10 - Create a signed certificate and key
     11 - Create a certificate renewal request

      0 - Exit program

Enter option number (press ENTER to return to previous menu): 6
```

Figure 9-36 Select export certificate to a file

```
Export File Format

      1 - Binary ASN.1 DER
      2 - Base64 ASN.1 DER
      3 - Binary PKCS #7
      4 - Base64 PKCS #7

Select export format (press ENTER to return to menu): 1
Enter export file name (press ENTER to return to menu): clientcert.der

Certificate exported.

Press ENTER to continue.
```

Figure 9-37 Select encoding format and file name then exit gskkyman

13. Copy the exported clientcert.der to a z/OS dataset such as LDAP8001.LDAP.CCERTB.

14. Add and connect LDAP8001.LDAP.CCERTB to the server's keyring file (in our example, LDAP8001.KEYRING):

```
RACDCERT ID(LDAP8001) ADD('LDAP8001.LDAP.CCERTB') TRUST WITHLABEL('LDAPCLT')

RACDCERT ID(LDAP8001) CONNECT(ID(LDAP8001) LABEL ('LDAPCLT')
RING(LDAP8001.KEYRING) USAGE(PERSONAL))
```

15. Stop the LDAP Server:

```
P LDAP8001
```

16. Open the configuration file PDS (in our example, LDAP8001.CNFOUT.DSCONFIG).

17. Edit the **sslAuth** parameter, and set it to **serverClientAuth** as shown in Figure 9-38.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
EDIT      LDAP8001.CNFOUT(DSCONFIG) - 01.05          Columns 00001 00072
Command ==>>>                               Scroll ==>>> CSR
000890 #-----
000891 # SSL/TLS specific CONFIGURATION SETTINGS
000892 #-----
000893
000894 #-----
000895 # sslAuth <serverAuth | serverClientAuth>
000896 #
000897 # Default Value: serverAuth
000898 #
000899 # Description:
000900 #   The sslAuth option specifies the SSL/TLS authentication method.
000901 #
000902 # Example:
000903 #   sslAuth serverClientAuth
000904 #-----
000905 sslAuth serverClientAuth
      F1=Help      F2=Split      F3=Exit      F5=Rfind      F6=Rchange      F7=Up
      F8=Down      F9=Swap      F10=Left     F11=Right     F12=Cancel
      . . . . .
```

Figure 9-38 Sample of DSCONFIG showing update for sslAuth

18. Start the LDAP server:

```
S LDAP8001
```

19. Test the client-server auth SSL using the **ldapsearch** command:

```
ldapsearch -h ldaphost -p sslport -D adminDN -w passwd -Z -K keyfile -P
keypasswd -s base -b "o=ibm,c=us" objectclass=*
```

You should see output similar to that seen in Example 9-1.

Example 9-1 ldapsearch command output

```
NPATEL:/u/npatel/sslkeys: >ldapsearch -D "cn=LDAP Admin, o=ibm, c=us" -w
sec001ret -p 4636 -Z -K "/u/npatel/sslkeys/ldapcltcert.kdb" -P "sec001ret" -s
base -b "o=ibm,c=us" objectclass=*
o=ibm,c=us
objectclass=top
objectclass=organization
o=ibm
```

9.11 Password policy implementation

Consider the following password rules defined by an example organization:

- ▶ Organization X has decided for security purposes they want their users to change their passwords every 28 days. Starting a week before a password expires, a user is to receive a warning when they log in. If a password expires, the administrator has to reset it. In addition, they decide a user can change their password as often as they like. There is no minimum time they must keep the same password.
- ▶ The organization realizes a user may be on vacation for two weeks and their password might expire during that time. It is decided that a user should be given one grace login after expiration to change their password. After that, if a user doesn't change their password, the admin has to reset it for them.
- ▶ This organization also wants the user to change their password immediately after the administrator resets it.
- ▶ Organization X has also decided that a user will only have three chances in a ten minute window to enter the correct password. If a user enters a bad password 3 times in ten minutes, they will be locked out. However, after an hour they can try again.
- ▶ When a user does change their password, the new password must be at least eight characters long, have six alphabetic characters, and two non-alphabetic characters. The password cannot have more than three of the same characters in a row. At least three characters should be different from the last password, and they cannot use their past two passwords. They must also provide their old password when they are changing their password for verification.
- ▶ There are five employees in the organization that belong to a group that only use the system occasionally, and it would be a pain if their passwords expired every month, so their passwords should only expire every 3 months. But they should follow all the other password policy rules.
- ▶ The organization's LDAP administrator, whose user ID and password are stored in the directory, doesn't need or want to adhere to any password policy rules such as expiration, lock out, or syntax rules for changing her password. It is decided that the administrator should be exempt from password policy.

How would this be accomplished? With a global password policy, and a group and individual policy.

The LDAP Server has a default global password policy, but by default it is disabled. To see the default after the server is first started, an **ldapsearch** can be performed:

```
ldapsearch -s base -b cn=pwdpolicy,cn=IBMpolicies objectclass=*
```

This command returns the output shown in Figure 9-39 on page 211.

```

cn=pwdpolicy,cn=ibmpolicies
objectclass=ibm-pwdgroupandindividualpolicies
objectclass=ibm-pwdPolicyExt
objectclass=pwdPolicy
objectclass=container
objectclass=top
pwdattribute=userpassword
pwdMinAge=0
pwdMaxAge=0
pwdInHistory=0
pwdCheckSyntax=0
pwdMinLength=0
pwdExpireWarning=0
pwdGraceLoginLimit=0
pwdLockout=false
pwdLockoutDuration=0
pwdMaxFailure=0
pwdFailureCountInterval=0
pwdMustChange=true
pwdAllowUserChange=TRUE
pwdSafeModify=FALSE
passwordMinAlphaChars=0
passwordMinOtherChars=0
passwordMaxRepeatedChars=0
passwordMinDiffChars=0
passwordMaxConsecutiveRepeatedChars=0
ibm-pwdPolicy=false
ibm-pwdGroupAndIndividualEnabled=false

```

Figure 9-39 Searching the default global password policy

To implement Organization X's password policy, the administrator would modify the global policy, define a group policy for those five employees in the occasional group, and define an individual policy for herself. She would then turn on the LDAP password policy.

Below are a sequence of operation that the administrator could do to implement the organizations requirements:

1. The administrator can perform an **ldapmodify** to modify the default global policy with the appropriate attribute values:

```
ldapmodify -D cn=admin,o=organizationX,c=ca -w secret -f aaa.ldif
```

The contents of the aaa.ldif file are shown in Figure 9-40.

```
dn: cn=pwdpolicy,cn=IBMpolicies
changetype:modify
replace:x
pwdMinAge: 0
pwdMaxAge: 2419200
pwdInHistory: 2
pwdCheckSyntax: 1
pwdMinLength: 8
pwdExpireWarning: 604800
pwdGraceLoginLimit: 1
pwdLockout: true
pwdLockoutDuration: 3600
pwdMaxFailure: 3
pwdFailureCountInterval: 600
pwdMustChange: true
pwdAllowUserChange: true
pwdSafeModify: true
passwordMinAlphaChars: 6
passwordMinOtherChars: 2
passwordMaxRepeatedChars: 0
passwordMinDiffChars: 3
passwordMaxConsecutiveRepeatedChars: 3
ibm-pwdGroupAndIndividualEnabled: true
```

Figure 9-40 Modifying the global policy

Note:

- ▶ Times are entered in seconds.
- ▶ **pwdmustchange** true and **pwdallowuserchange** false together are unacceptable.
- ▶ Although the global policy has enabled group and individual policies, those policies and the global policy will not take effect until the global policy is enabled. In other words password policy is not turned on yet. We will do that later.

2. The administrator can perform an **ldapmodify -a** to add a group password policy that overrides only the expiration time of the global policy but uses all the other values from the global policy. An **ldapmodify -a** for the group must also be done:

```
ldapmodify -D cn=admin,o=organizationX,c=ca -w secret -a -f aab.ldif
```

The contents of aab.ldif are shown in Figure 9-41.

```
dn: cn=ocassionalFolksPolicy,cn=IBMpolicies
objectclass: pwdPolicy
objectclass: ibm-pwdPolicyExt
objectclass: container
pwdattribute: userpassword
pwdMaxAge: 7776000
ibm-pwdPolicy: true
```

Figure 9-41 Adding a group policy and enabling it

For our example, adding the group with five members and a link to the group policy:

```
ldapmodify -D cn=admin,o=organizationX,c=ca -w secret -a -f aac.ldif
```

The contents of aac.ldif are shown in Figure 9-42.

```
dn: cn=GroupX,o=organizationX,c=ca
objectclass: groupOfNames
member: cn=Joe,o=organizationX,c=ca
member: cn=Sue,o=organizationX,c=ca
member: cn=Sally,o=organizationX,c=ca
member: cn=Fred,o=organizationX,c=ca
member: cn=Amanda,o=organizationX,c=ca
ibm-pwdGroupPolicyDN: cn=ocassionalFolksPolicy,cn=IBMpolicies
```

Figure 9-42 Adding a group with five members and a link to the group policy

Note: The policy is enabled, but will not take effect until the global policy is enabled. Defining a group and individual policy is done exactly the same way.

3. The administrator can perform an **ldapmodify** that exempts herself from having to follow any password policy rules:

```
ldapmodify -D cn=admin,o=organizationX,c=ca -w secret -f aad.ldif
```

The contents of aad.ldif are shown in Figure 9-43.

```
dn: cn=admin,o=organizationX,c=ca
changetype:modify
add:ibm-pwdIndividualPolicyDN
ibm-pwdIndividualPolicyDN: cn=noPwdPolicy
```

Figure 9-43 Modifying the Admin to be exempt from password policy with a special value

Note: **ibm-pwdIndividualPolicyDN** is used to assign an individual policy. In this case, a special policy name **cn=noPwdPolicy** is assigned. This special value is used to indicate that the user is exempt from all password policy checking.

4. Lastly, the administrator can perform an **ldapmodify** to enable the global password policy and the group and individual password policies. This turns on LDAP password policy.

```
ldapmodify -D cn=admin,o=organizationX,c=ca -w secret -f aae.ldif
```

The contents of aae.ldif are shown in Figure 9-44.

```
dn: cn=pwdpolicy,cn=IBMpolicies
changetype:modify
replace:x
ibm-pwdPolicy: true
```

Figure 9-44 Turning the global password policy on

5. And now search the existing policies:

```
ldapsearch -s sub -b cn=IBMpolicies objectclass=pwdPolicy
```

Sample output is shown in Figure 9-45.

```
cn=pwdpolicy,cn=ibmpolicies
objectclass=ibm-pwdgroupandindividualpolicies
objectclass=ibm-pwdPolicyExt
objectclass=pwdPolicy
objectclass=container
objectclass=top
pwdattribute=userPassword
pwdgraceloginlimit=1
pwsafemodify=true
pwdmaxfailure=3
pwdfailurecountinterval=600
pwdmaxage=2419200
pwdexpirerwarning=604800
pwdminlength=8
pwdlockout=true
pwdallowuserchange=true
pwdmustchange=true
ibm-pwdpolicy=true
ibm-pwdgroupandindividualenabled=true
passwordmaxconsecutivepeatedchars=3
passwordmaxrepeatedchars=0
passwordminalphachars=6
passwordminotherchars=2
passwordmindiffchars=3
pwdminage=0
pwdinhistory=2
pwdchecksyntax=1
cn=pwdpolicy
pwdlockoutduration=3600
ibm-pwdpolicystarttime=20100617190305.166676Z

cn=ocasionalFolksPolicy,cn=IBMpolicies
objectclass=pwdPolicy
objectclass=ibm-pwdPolicyExt
objectclass=container
objectclass=top
pwdattribute=userpassword
pwdmaxage=7776000
ibm-pwdpolicy=true
cn=ocasionalFolksPolicy
```

Figure 9-45 Search the existing policies

Remember there is no individual policy because **cn=noPwdPolicy** is just a special value.

There are a few extended operations and utilities that can help determine what is going on with password policy for a user or group and can facilitate in changing a user's password. The full syntax can be found in the *z/OS V1R12.0 IBM Tivoli Directory Server Administration and*

Use for z/OS, SC23-5191-05. Below are examples on how you might use them and, if appropriate, a sample result.

1. **ldapexop acctstatus**: An extended operation to show account status of open, locked, or expired:

```
ldapexop -D cn=admin,o=organizationX,c=ca -w secret -op acctstatus -d
cn=Laura,o=organizationX,c=ca
```

It will return one of the following:

- acctstatus_extended_op: Account is open.
- acctstatus_extended_op: Account is locked.
- acctstatus_extended_op: Account has expired.

2. **ldapexop effectpwdpolicy**: An extended operation to show the effective password policy, resolution of the combination of individual, group, and global policies.

```
ldapexop -D cn=admin,o=organizationX,c=ca -w secret -op effectpwdpolicy -d
cn=Laura,o=organizationX,c=ca
```

This operation produces the output shown in Figure 9-46.

```
The effective password policy is calculated based on the following entries
cn=pwdpolicy,cn=ibmpolicies

The effective password policy is:
ibm-pwdgroupandindividualenabled=TRUE
ibm-pwdpolicy=TRUE
ibm-pwdpolycystarttime=20100628200304.819480Z
passwordmaxconsecutiverepeatedchars=3
passwordmaxrepeatedchars=0
passwordminalphachars=6
passwordmindiffchars=3
passwordminotherchars=2
pwdallowuserchange=TRUE
pwdattribute=userpassword
pwdchecksyntax=1
pwdexpirewarning=604800
pwdfailurecountinterval=600
pwdgraceloginlimit=1
pwdinhistory=2
pwdlockout=TRUE
pwdlockoutduration=3600
pwdmaxage=2419200
pwdmaxfailure=3
pwdminage=0
pwdminlength=8
pwdmustchange=TRUE
pwsafemodify=TRUE
```

Figure 9-46 Output of ldapexop effectpwdpolicy

3. **ldapchangepwd** utility: Changes a user's password using current and new password:

```
ldapchangepwd -D cn=Laura,o=organizationX,c=ca -w secret -n rose1bud2
```

4. **ldapmodify** utility: The **ldapmodify** client utility can be used to change a password. Below is an example of how to change it, when **pwdSafeModify** is TRUE.

```
ldapmodify -D cn=Laura,o=organizationX,c=ca -w rose1bud2 -f change.ldif
```

The contents of change.ldif are shown in Figure 9-47.

```
dn: cn=Laura,o=organizationX,c=ca
changetype: modify
delete: userpassword
userpassword: rose1bud2
-
add: userpassword
userpassword: myword34
```

Figure 9-47 Change a password when pwdSafeModify is TRUE



Using IBM Tivoli Directory Server in a Parallel Sysplex

This chapter will show how to use IBM Tivoli Directory Server in a Parallel Sysplex environment. Changes to the LDBM and TDBM implementations that were done in the previous chapters will be shown.

10.1 Setting up the LDBM back end for sysplex

In this section, the changes that are needed for enhancing an existing LDBM instance to support sysplex will be shown.

The easiest way to set up IBM Tivoli Directory Server for sysplex is to use the same configuration files for each IBM Tivoli Directory Server instance and run IBM Tivoli Directory Server using the same started task name on each system in the sysplex. This guarantees that all instances are using the same configuration. If you decide to use a separate set of configuration options for each instance, ensure that the following options are set for each instance in the sysplex.

10.1.1 Changes to the configuration file

Specify the following options in the configuration file. In the example server, the changes are being made to the PDS that was created when `dsconfig` was run.

1. **serverSysplexGroup** LDAPLDB

The XCF group name specified must be the same for all servers sharing the same schema and back end database.

2. **database** LDBM GLDBLD31/GLDBLD64 LDBMDB1

3. **schemaPath** /u/ldapldb/schema

Each LDAP server in the XCF group must specify the same value for the **schemaPath** configuration option and must have read/write access to the specified directory or to `/var/ldap/schema` if the option is not specified. The schema directory used must exist within a shared z/OS UNIX System Services file system and must be accessible to all servers in the XCF group.

In the example setup, `/u` is an automounted directory that is shared across the sysplex. We created a z/FS dataset for LDAPTDBM that gets mounted automatically when the `/u/ldapldb` directory is referenced.

4. **multiserver** on

Specify this in each server's shared configuration file

5. Save the configuration file.

10.1.2 Starting and verifying operation

To verify the operation, perform the following steps:

1. Stop the LDAP instance if it is running:

```
P LDAD8001
P LDAP8101
```

2. Start the LDAP instances on each system using the modified configuration file in the sysplex that are part of the XCF group:

```
On System SC81 - S LDAD8101
On System SC80 - S LDAP8001
```

- Confirm that each LDAP instance has joined the XCF sharing group as shown in Example 10-1.

Example 10-1 XCF display

```

D XCF,GROUP,LDAPLDB,ALL
IXC333I 09.36.08 DISPLAY XCF 893
  INFORMATION FOR GROUP LDAPLDB
  MEMBER NAME:      SYSTEM:   JOB ID:   STATUS:
  SC8000BB          SC80      LDAP8001  ACTIVE
  SC8100B1          SC81      LDAP8101  ACTIVE

INFO FOR GROUP LDAPLDB MEMBER SC8000BB ON SYSTEM SC80

FUNCTION: Not Specified
MEMTOKEN: 01000017 002A0001      ASID: 00BB      SYSID: 01000110
  INFO: CURRENT      COLLECTED: 07/01/2010 09:36:08.621483

ATTRIBUTES      JOINED: 07/01/2010 09:33:36.776472
  JOIN TASK ASSOCIATION
  LOCAL CLEANUP NEEDED
  TERMLEVEL IS TASK
  MEMSTALL RESOLUTION IS NO ACTION
  EXITS DEFINED: MESSAGE, GROUP, NOTIFY

SIGNALLING SERVICE
MSGO ACCEPTED:      6 NOBUFFER:      0
MSGO XFER CNT:      8 LCL CNT:      0 BUFF LEN: 956
MSGO XFER CNT:      1 LCL CNT:      0 BUFF LEN: 53180
MSGO XFER CNT:      14 LCL CNT:      0 BUFF LEN: 62464

      SENDPND  RESPPND  COMPLTD  MOSAVED  MISAVED
MESSAGE TABLE:      0      0      0      0      0
  CRITICAL:      0      0      0      0      0

MSGI RECEIVED:      9 PENDINGQ:      0
MSGI XFER CNT:      8 XFERTIME:      N/A

      IO BUFFERS      DREF  PAGEABLE  CRITICAL
MSGI PENDINGQ:      0      0      0      0
SYMPATHY SICK:      0

GROUP SERVICE
  EVNT RECEIVED:      2 PENDINGQ:      0

EXIT 05D56190: 07/01/2010 09:33:41.952754 01 00:00:00.000003

INFO FOR GROUP LDAPLDB MEMBER SC8100B1 ON SYSTEM SC81

FUNCTION: Not Specified
MEMTOKEN: 02000012 002A0002      ASID: 00B1      SYSID: 0200010F
  INFO: CURRENT      COLLECTED: 07/01/2010 09:36:08.722777

ATTRIBUTES      JOINED: 07/01/2010 09:33:41.942851
  JOIN TASK ASSOCIATION
  LOCAL CLEANUP NEEDED
  TERMLEVEL IS TASK
  MEMSTALL RESOLUTION IS NO ACTION
  EXITS DEFINED: MESSAGE, GROUP, NOTIFY
SIGNALLING SERVICE

```

```

MSGO ACCEPTED:          6 NOBUFFER:          0
MSGO XFER CNT:         9  LCL CNT:          0  BUFF LEN:   956

                SENDPND  RESPPND  COMPLTD  MOSAVED  MISAVED
MESSAGE TABLE:      0      0      0      0      0
  CRITICAL:          0      0      0      0      0

MSGI RECEIVED:        6  PENDINGQ:          0
MSGI XFER CNT:       23  XFERTIME:          N/A

                IO BUFFERS      DREF  PAGEABLE  CRITICAL
MSGI PENDINGQ:        0          0      0          0
SYMPATHY SICK:        0

EXIT 0263F300: 07/01/2010 09:35:38.955125 DS 00:00:00.000024
EXIT 0263FB00: 07/01/2010 09:33:42.944653 NA 00:00:00.000061
EXIT 0263FD00: 07/01/2010 09:33:42.915742 NA 00:00:00.000032
EXIT 02640100: 07/01/2010 09:33:42.944793 OM 00:00:00.000241

GROUP SERVICE
  EVNT RECEIVED:      0  PENDINGQ:          0

```

10.2 Setting up the TDBM server for sysplex

In this section, the changes that are needed for enhancing an existing TDBM instance to support sysplex will be shown.

The easiest way to setup of IBM Tivoli Directory Server for sysplex is to use the same configuration files for each IBM Tivoli Directory Server instance and run IBM Tivoli Directory Server using the same started task name on each system in the sysplex. This guarantees that all instances are using the same configuration. If you decide to use a separate set of configuration options for each instance, ensure that the following options are set for each instance in the sysplex.

10.2.1 Changes to the configuration file

Specify the following options in the configuration file. In the example server, the changes are being made to the PDS that was created when **dsconfig** was run.

1. **serverSysplexGroup** LDAPXCF

The XCF group name specified must be the same for all servers sharing the same schema and back end database.

2. **database** TDBM GLDBTD31 GLDDB

For GLDDB, specify the name of the database that was identified in the DDL that was generated in your TDBSPUFI PDS member created by **dsconfig**. If you changed this name in your DDL, specify the name you used in this option.

3. **schemaPath** /u/ldaptdbm/schema

Each LDAP server in the XCF group must specify the same value for the **schemaPath** configuration option and must have read/write access to the specified directory or to **/var/ldap/schema** if the option is not specified. The schema directory used must exist

within a shared z/OS UNIX System Services file system and must be accessible to all servers in the XCF group.

In the example setup, /u is an automounted directory that is shared across the sysplex. We created a z/FS dataset for LDAPTDBM that gets mounted automatically when the /u/ldaptdbm directory is referenced.

Note about schema: Setting up the TDBM instance as described in the earlier chapter uses the default location for the schema of /var/ldap/schema. If you load the default schema, and then update the configuration to point to a new location for schema, you will get a mismatch on the schema definition and the database when you start your **ldap** instance:

```
GLD3301E Unable to load TDBM back end named GLDDB because attribute type
2.5.4.27 is not defined.
GLD1106E TDBM back end initialization failed for back end named GLDDB.
GLD1101A Unable to load the database back ends.
GLD1007I LDAP server is stopping.
```

To get around this problem, copy the schema directory from the original location to the location you have specified in your configuration and restart the **ldap** instance.

4. multiserver on

Specify this in each server's shared configuration file

5. Save the configuration file.

6. The GRANT statements in the DDL that were built and included in the PDS must be run for all LDAP servers participating in the datasharing TDBM group. Use SPUFI to execute the GRANT statements, usually located at the bottom of the DDL file.

10.2.2 Starting and verifying operation

To verify the operation, perform the following steps:

1. Stop the LDAP instance if it is running:

```
P LDAD8102
P LDAP8002
```

2. Start the LDAP instances on each system using the modified configuration file in the sysplex that are part of the XCF group:

```
On System SC81 - S LDAD8102
On System SC80 - S LDAP8002
```

3. Confirm that each LDAP instance has joined the XCF sharing group as shown in Example 10-2.

Example 10-2 XCF display

```
D XCF, GROUP, LDAPXCF, ALL
IXC333I 11.44.17 DISPLAY XCF 842
INFORMATION FOR GROUP LDAPXCF
MEMBER NAME:      SYSTEM:      JOB ID:      STATUS:
SC8000B7          SC80          LDAP8002    ACTIVE
SC8100BA          SC81          LDAP8102    ACTIVE

INFO FOR GROUP LDAPXCF MEMBER SC8000B7 ON SYSTEM SC80
```

FUNCTION: Not Specified
MEMTOKEN: 01000007 00290002 ASID: 00B7 SYSID: 01000110
INFO: CURRENT COLLECTED: 06/30/2010 11:44:17.103051

ATTRIBUTES JOINED: 06/30/2010 10:23:14.325369
JOIN TASK ASSOCIATION
LOCAL CLEANUP NEEDED
TERMLEVEL IS TASK
MEMSTALL RESOLUTION IS NO ACTION
EXITS DEFINED: MESSAGE, GROUP, NOTIFY

SIGNALLING SERVICE
MSGO ACCEPTED: 5 NOBUFFER: 0
MSGO XFER CNT: 8 LCL CNT: 0 BUFF LEN: 956

	SENDPND	RESPPND	COMPLTD	MOSAVED	MISAVED
MESSAGE TABLE:	0	0	0	0	0
CRITICAL:	0	0	0	0	0
MSGI RECEIVED:	5	PENDINGQ:		0	
MSGI XFER CNT:	22	XFERTIME:		N/A	

	IO BUFFERS	DREF	PAGEABLE	CRITICAL
MSGI PENDINGQ:	0	0	0	0
SYMPATHY SICK:	0			

EXIT 065AC700: 06/30/2010 11:44:07.298328 DS 00:00:00.000007

GROUP SERVICE
EVNT RECEIVED: 0 PENDINGQ: 0

INFO FOR GROUP LDAPXCF MEMBER SC8100BA ON SYSTEM SC81

FUNCTION: Not Specified
MEMTOKEN: 0200000B 00290001 ASID: 00BA SYSID: 0200010F
INFO: CURRENT COLLECTED: 06/30/2010 11:44:17.200326

ATTRIBUTES JOINED: 06/30/2010 10:04:39.937859
JOIN TASK ASSOCIATION
LOCAL CLEANUP NEEDED
TERMLEVEL IS TASK
MEMSTALL RESOLUTION IS NO ACTION
EXITS DEFINED: MESSAGE, GROUP, NOTIFY

SIGNALLING SERVICE
MSGO ACCEPTED: 15 NOBUFFER: 0
MSGO XFER CNT: 24 LCL CNT: 0 BUFF LEN: 956
MSGO XFER CNT: 42 LCL CNT: 0 BUFF LEN: 62464

	SENDPND	RESPPND	COMPLTD	MOSAVED	MISAVED
MESSAGE TABLE:	0	0	0	0	0
CRITICAL:	0	0	0	0	0

MSGI RECEIVED:	25	PENDINGQ:		0	
MSGI XFER CNT:	21	XFERTIME:		N/A	

	IO BUFFERS	DREF	PAGEABLE	CRITICAL
MSGI PENDINGQ:	0	0	0	0
SYMPATHY SICK:	0			

```
GROUP SERVICE
  EVNT RECEIVED:          8  PENDINGQ:          0

EXIT 01BBED50: 06/30/2010 10:23:14.334583 01 00:00:00.000005
```

10.3 Other shared back ends

This section describes how to enable sharing of CDBM and GDBM back ends to supplement the sharing of LDBM and TDBM back ends. Although they are not required to be shared, the user might find this to be useful.

Each LDBM, TDBM, CDBM, and GDBM back end can be shared within the XCF group. To share a back end, specify `multiserver` in the back end section in the configuration file of each LDAP server. If `multiserver` off is specified or if the `multiserver` option is not specified, the back end is not shared and changes to the back end are not reflected in the other servers on the sysplex, even if they contain the same suffix. If GDBM or CDBM back ends are configured, all LDBM, TDBM, CDBM, and GDBM back ends must be shared or not shared. However, if GDBM and CDBM back ends are not configured, some LDBM and TDBM back ends can be shared while others are not shared.

10.4 Setup a shared GDBM to track changes

This section shows how to setup GDBM to track changes to the server database. To enable GDBM for an LDBM IBM Tivoli Directory Server instance, do the following:

1. Stop the LDAP server:

```
P LDAP8001
P LDAP8101
```

2. Open `LDAP8001.CNFOUT(DSCONFIG)`.
3. If not already specifying the database to use for GDBM, uncomment the line `database GDBM GLDBGD31/GLDBGD64` and add the name of the database. In the example shown in Figure 10-1 on page 224, the database name is `GDBML`.

Note: Do not uncomment the line `database GDBM GLDBGD31` because it is used for the DB2-based GDBM back end section of the configuration file.

```

EDIT          LDAP8001.CNFOUT(DSCONFIG) - 01.01          Columns 00001 00072
Command ==>          Scroll ==> CSR
003669 #-----
003670 # database dbtype dblibpath
003671 #
003672 # Description:
003673 #   The database option marks the beginning of a new database section.
003674 #
003675 # Example:
003676 #   database GDBM GLDBGD31/GLDBGD64
003677 #
003678 # Notes:
003679 #   All global options must appear before the first database section.
003680 #   An optional name may be specified to identify this back end.
003681 #-----
003682 database GDBM GLDBGD31/GLDBGD64 GDBML
003683
003684 #-----

```

Figure 10-1 DSCONFIG example of database GDBM

4. If you have not already specified the database directory for the GDBM database, uncomment the GDBM-specific **databaseDirectory** option, where IBM Tivoli Directory Server server will create GDBM / change log related files. If the specified directory is not available, then IBM Tivoli Directory Server instance will create one. See Figure 10-2.

```

EDIT          LDAP8001.CNFOUT(DSCONFIG) - 01.02          Columns 00001 00072
Command ==>          Scroll ==> CSR
003815
003816 #-----
003817 # databaseDirectory <name>
003818 #
003819 # Default Value: /var/ldap/gdbm
003820 #
003821 # Description:
003822 #   The databaseDirectory option specifies the name of the file system
003823 #   directory containing the data files used by this back end, to store
003824 #   the directory data.
003825 #
003826 # Example:
003827 #   databaseDirectory /home/myLdap/gdbmData
003828 #
003829 # Notes:
003830 #   A fully-qualified directory path must be specified. A unique file
003831 #   system directory must be specified for each file-based back end.
003832 #   In addition, when multi-server mode is active, the same directory
003833 #   path must be specified for each instance of this back end within the
003834 #   cross-system group.
003835 #-----
003836 databaseDirectory /u/ldapldb/gdbm
003837
003838 #-----

```

Figure 10-2 DSCONFIG example of databaseDirectory

5. Specify **multiserver on** in the GDBM-specific section
6. Save the configuration file.

7. Start the LDAP server:

```
S LDAP8001
GLD1004I LDAP server is ready for requests.
```

8. LDAP will create the directory specified in the **databaseDirectory** configuration option. However, the permissions may not be correct to allow read/write to all servers that will be accessing the directory. In this example, the **gdbm** directory was created with permissions set to 740, so we changed this to 777:

```
ls -l
total 6
drwxr----- 2 LDAP8001 LDAPGRP      352 Jul  1 10:03 gdbm
drwxrwx--- 2 LDAP8001 LDAPGRP      384 Jul  1 10:03 ldbm
drwxrwx--- 2 LDAP8001 LDAPGRP      320 Jun 30 15:20 schema
```

Update the permissions on the **gdbm** directory to 777:

```
chmod 770 gdbm
ls -l /u/ldap1dbm
total 6
drwxrwx--- 2 LDAP8001 LDAPGRP      352 Jul  1 10:03 gdbm
drwxrwx--- 2 LDAP8001 LDAPGRP      384 Jul  1 10:03 ldbm
drwxrwx--- 2 LDAP8001 LDAPGRP      320 Jun 30 15:20 schema
```

9. The files within the **gdbm** directory may not be created with permissions set to allow read/write to all servers. Change the permissions to 660:

```
ls -l
total 4
-rw-r----- 1 LDAP8001 LDAPGRP      706 Jul  1 10:03 LDBM-1.db
-rw-r----- 1 LDAP8001 LDAPGRP       35 Jul  1 10:03 LDBM.ckpt
chmod 660 *
ls -l
total 4
-rw-rw---- 1 LDAP8001 LDAPGRP      706 Jul  1 10:03 LDBM-1.db
-rw-rw---- 1 LDAP8001 LDAPGRP       35 Jul  1 10:03 LDBM.ckpt
```

10. Start the other LDAP servers in the same sysplex group to pick up the changes.

10.5 Set up a shared CDBM for advanced replication and password policy

This section shows how to set up CDBM for advanced replication functions. To enable CDBM for an LDBM IBM Tivoli Directory Server instance, perform the following steps:

1. Stop the LDAP server:

```
P LDAP8001
P LDAP8101
```

2. Open LDAP8001.CNFOUT(DSCONFIG).

- If not already specifying the database to use for CDBM, uncomment the line `database CDBM GLDBCD31/GLDBCD64` and add the name of the database. In the example shown in Figure 10-3, the database name is **cdbm**.

```

Command ==>                               Scroll ==> PAGE
003021
003022 # CDBM-specific CONFIGURATION SETTINGS
003023
003024 #-----
003025 #-----
003026 #-----
003027 #-----
003028
003029
003030 #-----
003031 # database dbtype dblibpath
003032 #
003033 # Description:
003034 #   The database option marks the beginning of a new database section.
003035 #
003036 # Example:
003037 #   database CDBM GLDBCD31/GLDBCD64
003038 #
003039 # Notes:
003040 #   All global options must appear before the first database section.
003041 #   An optional name may be specified to identify this back end.
003042 #-----
003043 database CDBM GLDBCD31/GLDBCD64 cdbm

```

Figure 10-3 Adjusting `LDAP8001.CNFOUT(DSCONFIG)`

- If not already specifying the database directory for the CDBM database, uncomment the CDBM specific `databaseDirectory` configuration option as shown in Figure 10-4.

```

EDIT      LDAP8001.CNFOUT(DSCONFIG) - 01.30          Columns 00001 00072
Command ==>                               Scroll ==> CSR
003134 #-----
003135 # databaseDirectory <name>
003136 #
003137 # Default Value: the value of schemaPath
003138 #
003139 # Description:
003140 #   The databaseDirectory option specifies the name of the file system
003141 #   directory containing the data files used by this back end, to store
003142 #   the directory data.
003143 #
003144 # Example:
003145 #   databaseDirectory /home/myCDBM
003146 #
003147 # Notes:
003148 #   A fully-qualified directory path must be specified. It is
003149 #   recommended that this option is set to the same value as
003150 #   schemaPath. The LDAP server will set this value to the value of
003151 #   schemaPath when this value is not explicitly set.
003152 #-----
003153 # databaseDirectory /u/1dap1dbm/cdbm

```

Figure 10-4 Uncommenting the `databaseDirectory` configuration option

5. Specify `multiserver on` in the CDBM-specific section.
6. Save the configuration file.
7. Start the LDAP server:

```
S LDAP8001
GLD1004I LDAP server is ready for requests.
```

8. LDAP will create the directory specified in the `databaseDirectory` configuration option. However, the permissions may not be correct to allow read/write to all servers that will be accessing the directory. In this example, the `cdm` directory was created with permissions set to 740, so we changed them to 777:

```
ls -l
total 8
drwxr----- 2 LDAP8001 LDAPGRP      352 Jul  1 11:46 cdbm
drwxrwx--- 2 LDAP8001 LDAPGRP      384 Jul  1 11:46 gdbm
drwxrwx--- 2 LDAP8001 LDAPGRP      384 Jul  1 11:46 ldbm
drwxrwx--- 2 LDAP8001 LDAPGRP      320 Jun 30 15:20 schema
```

Update the permissions on the `cdm` directory to 777:

```
chmod 770 cdbm
ls -l /u/ldap/dbm
total 8
drwxrwx--- 2 LDAP8001 LDAPGRP      352 Jul  1 11:46 cdbm
drwxrwx--- 2 LDAP8001 LDAPGRP      384 Jul  1 11:46 gdbm
drwxrwx--- 2 LDAP8001 LDAPGRP      384 Jul  1 11:46 ldbm
drwxrwx--- 2 LDAP8001 LDAPGRP      320 Jun 30 15:20 schema
```

9. The files within the `cdm` directory may not be created with permissions set to allow read/write to all servers. Change the permissions to 660:

```
ls -l
total 12
-rw-r----- 1 LDAP8001 LDAPGRP      37 Jul  1 11:46 LDBM-1.db
-rw-r----- 1 LDAP8001 LDAPGRP      35 Jul  1 11:46 LDBM-2.db
-rw-r----- 1 LDAP8001 LDAPGRP    3330 Jul  1 11:46 LDBM.ckpt
chmod 660 *
ls -l
total 12
-rw-rw---- 1 LDAP8001 LDAPGRP      37 Jul  1 11:46 LDBM-1.db
-rw-rw---- 1 LDAP8001 LDAPGRP      35 Jul  1 11:46 LDBM-2.db
-rw-rw---- 1 LDAP8001 LDAPGRP    3330 Jul  1 11:46 LDBM.ckpt
```

10. Make the same modifications to the configuration files for the other servers in the same `sysplex` group.

Note: If GDBM or CDBM back ends are configured, then all LDBM, TDBM, CDBM and GDBM back ends must be shared, or they all must be not shared.

11. Start the other LDAP servers in the same `sysplex` group to pick up the changes.



Replication

Replication is a technique used by directory servers to improve performance, availability, and reliability. The replication process keeps the data in multiple directory servers synchronized. Through replication, a change made to one directory is propagated to one or more additional directories. In effect, a change to one directory is propagated to multiple directories.

Replication provides three main benefits:

- ▶ Redundancy of information: Replicas back up the content of their supplier servers.
- ▶ Faster searches: Search requests can be spread among several servers instead of a single server. This improves the response time for request completion.
- ▶ Security and content filtering: Replicas can contain subsets of the data in a supplier server.

In z/OS IBM Tivoli Directory Server, we can have two types of replication:

1. Basic replication
2. Advanced replication

This chapter provides sample configuration of both replication types.

11.1 Basic Replication

Basic replication is simple to configure, but it has limitations:

- ▶ Nested replication is not allowed.
- ▶ Subtree-based replication is not allowed.
- ▶ Scheduled replication is not allowed.
- ▶ There is no way to monitor configured replication.
- ▶ Partial replication is not allowed.

Note: Basic replication is not allowed if CDBM is configured.

You can configure several topologies using basic replication, which are covered in the following sections.

11.1.1 Master - replica topology

The simplest replication topology for basic replication is that of a master server and its replica server. The master server can contain a directory or a subtree of a directory. The master is writable, which means it can receive updates from clients for a given subtree. The replica server contains a copy of the directory of the master server. The replica is read only; it cannot be directly updated by clients. Instead it refers client requests to the master server, which then performs the updates and replicates them to the replica server.

A master server can have several replicas. Each replica can contain a copy of the master's entire directory.

Master-Replica replication topology configuration in basic replication

Configuring a simple master-replica scenario involves the following steps:

1. Define replication configuration parameters:
 - a. Master server's host name or IP address and the port on which it is listening
 - b. Replica server's host name or IP address and the port on which it is listening
 - c. Replicating suffix
 - d. Secure or non-secure replication communication between master and replica server
 - e. Replication bind DN, which the master uses as a bind DN while replicating changes to replica
 - f. Replication bind credentials, which is the password of the replication bind DN
 - g. Replication log file name and location

In our replication configuration example, we used the following parameters:

- a. Master server is listening on WTSC80.ITS0.IBM.COM: 13389.
- b. Replica is listening on WTSC81.ITS0.IBM.COM: 13389.
- c. Replication starting point would be `o=ibm,c=us`, which is configured as a suffix in the LDBM back end.
- d. Non-SSL based replication is being configured.
- e. Replication bind DN is `cn=Master`.

- f. Replication bind credential is secret.
- g. Replication log file name and location is /u/ldap8001/logs/replicaldap8101.errlog.

As per the requirements, one can edit the configuration parameters such as master and replica host name, port, replica bind DN, and replica bind password.

2. Create an ldif file on the master server, name it cred.ldif, and copy the following contents into it:

```
dn: cn=Replicaldap8101,o=ibm,c=us
objectclass: replicaObject
objectclass: extensibleObject
cn: Replicaldap8101
replicaHost: WTSC81.ITS0.IBM.COM
replicaBindDn: cn=Master
replicaCredentials: secret
replicaPort: 13389
replicaUseSSL: FALSE
description: Replica server
ibm-slapdLog: /u/ldap8001/logs/replicaldap8101.errlog
```

Note: Do not to use cn=Master and secret as your replica bind DN and credential. Instead, change them to meet your requirements.

3. Load the cred.ldif file on master server. Use the **ldapadd** command to load the file:

```
ldapadd -h ldaphost -p port -D adminDN -w passwd -f cred.ldif
```

In our example:

```
ldapadd -h WTSC80.ITS0.IBM.COM -p 13389 -D cn=root -w root -f cred.ldif
```

4. Switch to the replica server.
5. Open the configuration file PDS. The following example uses the file LDAP8101.CNFOUT (DSCONFIG).

6. Edit the **masterServer** parameter of LDBM configuration stanza to the master server's IP/host name and port. These values must match with the values defined above, and host name and port should be separated by a colon (:) as shown in Figure 11-1.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      LDAP8101.CNFOUT(DSCONFIG) - 01.00          Columns 00001 00072
Command ==>                                     Scroll ==> CSR
002649 #-----
002650 # masterServer <ldap-url>
002651 #
002652 # Description:
002653 #   The masterServer option specifies the location of this replica's
002654 #   master server, for this back end, for basic replication. The
002655 #   presence of this option indicates that this LDAP server is a basic
002656 #   replication read-only replica for this back end and receives updates
002657 #   from a master LDAP server. Any other update requests for this
002658 #   back end received directly by the this LDAP server is redirected to
002659 #   the master server.
002660 #
002661 # Example:
002662 #   masterServer ldap://ldbMaster.server.com:3389
002663 #
002664 # Notes:
002665 #   The masterServerDN option must also be specified in this section of
002666 #   the configuration file. The masterServer option indicates basic
002667 #   replication is configured for this back end section. The
002668 #   masterServer option cannot be specified if the
002669 #   useAdvancedReplication option is set to 'on' in the CDBM back end
002670 #   database section.
002671 #-----
002672 masterServer ldap://WTSC80.ITS0.IBM.COM:13389
002673
```

Figure 11-1 Sample DSCONFIG file showing update for masterServer

7. Edit the `masterServerDN` parameter of LDBM configuration to the value of the `replicaBindDn` attribute is used in step 1 (Figure 11-2).

Note: Do not use `adminDN` as `masterServerDN`. This might cause unforeseen problems.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      LDAP8101.CNFOUT(DSCONFIG) - 01.00          Columns 00001 00072
Command ==>                               Scroll ==> CSR
002684 # replicating to this read-only replica back end. The DN has
002685 # unrestricted update, compare, and search access for all entries in
002686 # the back end on this server, even if the LDAP server is in
002687 # maintenance mode. When in maintenance mode, only this DN and the
002688 # LDAP administrator can access and update the entries in this
002689 # back end. All other update operations for this back end received by
002690 # the replica server are redirected to the master server. Care must
002691 # be taken when updating this back end to ensure the replica server
002692 # remains synchronized with the master server.
002693 #
002694 # Example:
002695 # masterServerDN "cn=Master Server, o=Your Company"
002696 #
002697 # Notes:
002698 # You must also specify the masterServer option in this section of
002699 # the configuration file. You cannot specify the peerServerDN
002700 # option.
002701 #
002702 # The masterServerDN option indicates basic replication is configured
002703 # for this back end section. Therefore, the masterServerDN option
002704 # cannot be specified if the useAdvancedReplication option is set to
002705 # 'on' in the CDBM back end database section
002706 #-----
002707 masterServerDN "cn=Master"
002708
```

Figure 11-2 Sample DSCONFIG showing update for `masterServerDN`

replicas and to the other peer masters, but do not replicate updates received from other master servers.

Peer to peer replication topology configuration in basic replication.

Configuring a peer to peer scenario involves the following steps:

1. Define the following given replication configuration parameters:
 - a. Peer1 server's host name or IP address and the port on which it is listening
 - b. Peer2 server's host name or IP address and the port on which it is listening
 - c. Replicating suffix
 - d. Secure or non-secure replication communication between peer1 and peer2 server
 - e. Replication bind DN, which the peer uses as a bind DN while replicating changes to another peer
 - f. Replication bind credentials, which is the password of the replication bind DN
 - g. Replication log file name and location

In the replication configuration example, we have used the following parameters:

- a. Peer1 server is listening on WTSC80.ITS0.IBM.COM: 14389.
- b. Peer2 is listening on WTSC81.ITS0.IBM.COM: 14389.
- c. Replication starting point would be `o=ibm,c=us`, which is configured as a suffix in the LDBM back end.
- d. Non-SSL based replication is being configured.
- e. Replication bind DN is `cn=Master`.
- f. Replication bind credential is `secret`.
- g. Replication log file names and locations are:
`/u/ldap8002/logs/replicaldap8102.errlog`
`/u/ldap8102/logs/replicaldap8002.errlog`

As per the requirements, one can edit the configuration parameters such as master and replica host name, port, replica bind DN, and replica bind password.

2. Create an `ldif` file on the peer1 server, name it `peer1.ldif`, and copy the following contents into it:

```
dn: cn=Replicaldap8102,o=ibm,c=us
objectclass: replicaObject
objectclass: extensibleObject
cn: Replicaldap8102
replicaHost: WTSC81.ITS0.IBM.COM
replicaBindDn: cn=Master
replicaCredentials: secret
replicaPort: 14389
replicaUseSSL: FALSE
description: Replica server
ibm-slapdLog: /u/ldap8002/logs/replicaldap8102.errlog
```

Note: Do not use `cn=Master` and `secret` as your replica bind DN and credential. Change them to meet your requirements.

7. Edit the **peerServerPW** parameter of LDBM configuration stanza to match the value of **replicaCredentials** attribute used in step 1 (Figure 11-5).

```

File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      LDAP8002.CNFOUT(DSCONFIG) - 01.00          Columns 00001 00072
Command ==>                               Scroll ==> CSR
002837 # peerServerPW <password>
002838 #
002839 # Description:
002840 #   The peerServerPW option specifies the password for the
002841 #   peerServerDN that is allowed to make updates for this back end.
002842 #   This option is only applicable for a basic replication peer
002843 #   replica LDAP server.
002844 #
002845 # Example:
002846 #   peerServerPW pPassword
002847 #
002848 # Notes:
002849 #   Use of the peerServerPW configuration option is strongly
002850 #   discouraged in production environments. Instead, specify your
002851 #   peerServerDN as the distinguished name of an existing entry in the
002852 #   directory information tree, including a userPassword attribute.
002853 #   This will eliminate passwords from the configuration file.
002854 #
002855 #   The peerServerPW option indicates basic peer-to-peer replication
002856 #   is configured for this back end section. Therefore, the
002857 #   peerServerPW option cannot be specified if the
002858 #   useAdvancedReplication option is set to 'on' in the CDBM back end
002859 #   database section.
002860 #-----
002861 peerServerPW secret
002862
F1=Help      F2=Split    F3=Exit     F5=Rfind    F6=Rchange  F7=Up
F8=Down      F9=Swap     F10=Left    F11=Right   F12=Cancel

```

Figure 11-5 Sample of showing update for peerServerPW

8. Save the configuration file PDS. Our example uses the file LDAP8002.CNFOUT(DSCONFIG).
9. Switch to the peer2 server.
10. Create an ldif file on the peer2 server, name it peer2.ldif, and copy the following contents into it:

```

dn: cn=ReplicaLdap8002,o=ibm,c=us
objectclass: replicaObject
objectclass: extensibleObject
cn: ReplicaLdap8002
replicaHost: WTSC80.ITS0.IBM.COM
replicaBindDn: cn=Master
replicaCredentials: secret
replicaPort: 14389
replicaUseSSL: FALSE
description: Replica server
ibm-slapdLog: /u/ldap8102/logs/repicaldap8002.errlogS

```


15. Edit the **peerServerPW** parameter of LDBM configuration parameter to match the value of **replicaCredentials** attribute used in step 10 (Figure 11-7).

```

File Edit Edit_Settings Menu Utilities Compilers Test Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
EDIT      LDAP8102.CNFOUT(DSCONFIG) - 01.02                Columns 00001 00072
Command ==>                                           Scroll ==> CSR
002837 # peerServerPW <password>
002838 #
002839 # Description:
002840 #   The peerServerPW option specifies the password for the
002841 #   peerServerDN that is allowed to make updates for this back end.
002842 #   This option is only applicable for a basic replication peer
002843 #   replica LDAP server.
002844 #
002845 # Example:
002846 #   peerServerPW pPassword
002847 #
002848 # Notes:
002849 #   Use of the peerServerPW configuration option is strongly
002850 #   discouraged in production environments. Instead, specify your
002851 #   peerServerDN as the distinguished name of an existing entry in the
002852 #   directory information tree, including a userPassword attribute.
002853 #   This will eliminate passwords from the configuration file.
002854 #
002855 #   The peerServerPW option indicates basic peer-to-peer replication
002856 #   is configured for this back end section. Therefore, the
002857 #   peerServerPW option cannot be specified if the
002858 #   useAdvancedReplication option is set to 'on' in the CDBM back end
002859 #   database section.
002860 #-----
002861 peerServerPW secret
    F1=Help      F2=Split      F3=Exit      F5=Rfind      F6=Rchange    F7=Up
    F8=Down      F9=Swap       F10=Left     F11=Right     F12=Cancel
. . . . .

```

Figure 11-7 Sample of DSCONFIG showing update for **peerServerPW**

16. Save the configuration file PDS. Our example uses the file LDAP8102.CNFOUT(DSCONFIG).

17. Restart peer1 server:

```

P LDAP8002
S LDAP8002

```

18. Restart peer2 server:

```

P LDAP8102
S LDAP8102

```

Note: Use **secretEncryption** to encrypt your replica bind password in the conf file and in the back end (if it is configured).

11.2 Advanced Replication

To overcome the limitations of basic replication, advanced replication was added in the z/OS IBM Tivoli Directory Server server with the CDBM back end. This advanced replication feature provides:

- ▶ Subtree based replication
- ▶ Nested replication
- ▶ Separate roles for different subtrees of the single server in the replication
- ▶ Additional replication topologies
- ▶ External error log management using extended operations
- ▶ External replication queue management using extended operations
- ▶ New operational attributes to determine the status of replication
- ▶ Schema replication
- ▶ Partial replication
- ▶ Scheduled replication
- ▶ Replication conflict resolution

11.2.1 Major replication topologies

In z/OS IBM Tivoli Directory Server using advance replication one can have the following replication topologies:

Master - replica topology

The basic relationship in advanced replication is that of a master server and its replica server. The master server can contain a directory or a subtree of a directory. The master is writable, which means it can receive updates from clients for a given subtree. The replica server contains a copy of the directory or a copy of part of the directory of the master server. The replica is read only: it cannot be directly updated by clients. Instead it refers client requests to the master server, which then performs the updates and replicates them to the replica server.

A master server can have several replicas. Each replica can contain a copy of the master's entire directory, or a subtree of the directory.

Peer to peer topology

There can be several servers acting as masters for directory information, with each master responsible for updating other master servers and replica servers. This is referred to as peer replication. Peer replication can improve performance, availability, and reliability. Performance is improved by providing a local server to handle updates in a widely distributed network. Availability and reliability are improved by providing a backup master server ready to take over immediately if the primary master fails. Peer master servers replicate all client updates to the replicas and to the other peer masters, but do not replicate updates received from other master servers.

Cascading replication topology

Forwarding (cascading) replication is a topology that has multiple tiers of servers. A master server replicates to a set of read-only (forwarding) servers that in turn replicate to other servers. Such a topology off-loads replication work from the master server.

Gateway replication topology

Gateway replication is a more complex adaptation of peer-to-peer replication that extends replication capabilities across networks. Gateway peer servers will replicate updates from other Gateway peer servers to all servers in their sub-network only. On the other hand, a

Gateway will replicate to other gateway peers any updates received from clients or servers within its sub-network.

11.2.2 Configuring replication topologies

Before starting configuration of advanced replication, you will need to configure a CDBM back end. Follow these steps to enable advanced replication:

1. Stop the LDAP server, if it is running. In the following example, the instance name is LDAP8001.

```
P LDAP8001
```

You should receive confirmation like the following:

```
GLD1006I LDAP server stop command received.  
BPXM023I (LDAP8001) GLD1007I LDAP server is stopping.  
BPXM023I (LDAP8001) GLD6051I No database changes to commit for GDBM back end  
named GDBM-0001.  
BPXM023I (LDAP8001) GLD6051I No database changes to commit for LDBM back end  
named LDBM-0002.  
$HASP395 LDAP8001 ENDED
```

2. Open the configuration file PDS. Our example uses the file LDAP8001.CNFOUT(DSCONFIG).
3. Enable CDBM back end by uncommenting the line database CDBM GLDBCD31/GLDBCD64 cdbm as shown in Figure 11-8.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help  
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss  
EDIT          LDAP8001.CNFOUT(DSCONFIG) - 01.06          Columns 00001 00072  
Command ==>>>                                     Scroll ==>> CSR  
003029  
003030 #-----  
003031 # database dbtype dblibpath  
003032 #  
003033 # Description:  
003034 #   The database option marks the beginning of a new database section.  
003035 #  
003036 # Example:  
003037 #   database CDBM GLDBCD31/GLDBCD64  
003038 #  
003039 # Notes:  
003040 #   All global options must appear before the first database section.  
003041 #   An optional name may be specified to identify this back end.  
003042 #-----  
003043 database CDBM GLDBCD31/GLDBCD64 cdbm  
003044  
003045 #-----  
003046 # include <filename>  
F1=Help      F2=Split    F3=Exit      F5=Rfind     F6=Rchange   F7=Up  
F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Figure 11-8 Sample of DSCONFIG showing update for database “CDBM GLDBCD31/GLDBCD64 cdbm”

4. Edit **databaseDirectory** and provide a location where you would like store CDBM database files (Figure 11-9).

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
EDIT      LDAP8001.CNFOUT(DSCONFIG) - 01.07          Member DSCONFIG saved
Command ===>                                       Scroll ===> CSR
003134 #-----
003135 # databaseDirectory <name>
003136 #
003137 # Default Value: the value of schemaPath
003138 #
003139 # Description:
003140 #   The databaseDirectory option specifies the name of the file system
003141 #   directory containing the data files used by this back end, to store
003142 #   the directory data.
003143 #
003144 # Example:
003145 #   databaseDirectory /home/myCDBM
003146 #
003147 # Notes:
003148 #   A fully-qualified directory path must be specified. It is
003149 #   recommended that this option is set to the same value as
003150 #   schemaPath. The LDAP server will set this value to the value of
003151 #   schemaPath when this value is not explicitly set.
003152 #-----
003153 databaseDirectory /u/ldap8001/cdbm
F1=Help      F2=Split    F3=Exit     F5=Rfind    F6=Rchange  F7=Up
F8=Down      F9=Swap     F10=Left   F11=Right   F12=Cancel
```

Figure 11-9 Sample of DSCONFIG showing update for databaseDirectory

Note: The server will create the directory specified in the **databaseDirectory** option if the LDAP server's user ID has read/write permission to the directory and any parent directories.

5. Edit the `useAdvancedReplication` parameter and enable it for advanced replication as shown in Figure 11-10.

```

File Edit Edit_Settings Menu Utilities Compilers Test Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
EDIT      LDAP8001.CNFOUT(DSCONFIG) - 01.11                Columns 00001 00072
Command ==>                                           Scroll ==> CSR
003595 # -----
003596 # useAdvancedReplication <on | off>
003597 #
003598 # Default Value: off
003599 #
003600 # Description:
003601 #   The useAdvancedReplication option specifies if the LDAP server
003602 #   supports advanced replication. If advanced replication is active,
003603 #   then the masterServer, masterServerDN, masterServerPW, peerServer,
003604 #   peerServerDN, and peerServerPW configuration options cannot be
003605 #   specified in any LDBM, TDBM, or CDBM back ends.
003606 #
003607 # Example:
003608 #   useAdvancedReplication on
003609 #
003610 # Notes:
003611 #   The LDAP server will not start when useAdvancedReplication on is
003612 #   specified and entries with an objectclass of replicaObject are
003613 #   present in a TDBM, LDBM, or CDBM back end.
003614 #
003615 #   The LDAP server will not start when useAdvancedReplication off is
003616 #   specified and entries with an auxiliary objectclass of
003617 #   ibm-replicationContext are present in a TDBM, LDBM, or CDBM
003618 #   back end.
003619 #
003620 #   The server compatibility level must be at least 5.
003621 # -----
003622 useAdvancedReplication on
003623
F1=Help      F2=Split    F3=Exit     F5=Rfind    F6=Rchange  F7=Up
F8=Down      F9=Swap     F10=Left    F11=Right   F12=Cancel

```

Figure 11-10 Sample of DSCONFIG showing update for useAdvancedReplication

6. Save and close the configuration file.

Note: If CDBM is already configured, you do not need to modify the database `CDBM GLDBCD31/GLDBCD64` and `databaseDirectory` parameters.

7. Start the LDAP server:

```
S LDAP8001
```

The following output should appear:

```

S LDAP8001
$HASP100 LDAP8001 ON STCINRDR
IEF695I START LDAP8001 WITH JOBNAME LDAP8001 IS ASSIGNED TO USER LDAP8001,
GROUP LDAPGRP
$HASP373 LDAP8001 STARTED
BPXM023I (LDAP8001) GLD1004I LDAP server is ready for requests.
GLD1005I LDAP server start command processed.

```

```
BPXM023I (LDAP8001) GLD1059I Listening for requests on 9.12.4.45 port 4389.
BPXM023I (LDAP8001) GLD1059I Listening for requests on 9.12.4.46 port 4389.
BPXM023I (LDAP8001) GLD1059I Listening for requests on 9.12.5.26 port 4389.
BPXM023I (LDAP8001) GLD1059I Listening for requests on 127.0.0.1 port 4389.
```

Note: The LDAP server will fail to start after enabling CDBM if entries with an **objectclass** of `replicaObject` are present in a TDBM, LDBM, or CDBM back end.

8. The **cn=configuration** suffix contains entries that are used to configure advanced replication support. When the server is first started, the following advanced replication configuration entries under the **cn=configuration** suffix are automatically created:

```
cn=configuration
cn=Replication,cn=configuration
cn=Log Management,cn=Configuration
cn=Replication,cn=Log Management,cn=Configuration
```

Use the **ldapsearch** command to retrieve **cn=configuration** entries:

```
ldapsearch -h ldaphost -p port -D adminDN -w adminpw -s sub -b
"cn=configuration" objectclass=*
```

For example:

```
ldapsearch -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -p 4389 -s sub -b
"cn=configuration" objectclass=*
```

You should receive the following output:

```
cn=configuration
objectclass=ibm-tdszTop
objectclass=ibm-slapedConfigEntry
objectclass=top
cn=configuration
ibm-slapedserverid=134CF000-1D9B-1C25-9F09-402817013BD5
```

```
cn=Replication,cn=configuration
objectclass=ibm-slapedReplicationConfiguration
objectclass=top
cn=Replication
ibm-slapedmaxpendingchangesdisplayed=200
ibm-slapedreplcontextcachesize=100000
ibm-slapedreplmaxerrors=0
ibm-slapedreplconflictmaxentrysize=0
ibm-replicationonhold=FALSE
```

```
cn=Log Management,cn=Configuration
objectclass=container
objectclass=top
cn=Log Management
```

```
cn=Replication,cn=Log Management,cn=Configuration
objectclass=ibm-slapedLogConfig
objectclass=ibm-slapedConfigEntry
objectclass=container
objectclass=top
cn=Replication
ibm-slapedlog=/var/ldap/logs/lostfound.log
```

11.2.3 Master-Replica replication configuration in advanced replication.

Configuring a simple master-replica scenario involves the following steps:

1. Define the configuration parameters:
 - a. Replication context: This entry will be the starting point of the replication, and can be located anywhere in the tree. It is not necessary for it to be a suffix as in basic replication. In the following configuration example it is `o=ibm,c=us`.
 - b. Supplier(s): The LDAP server on `WTSC80.ITSO.IBM.COM:13389` will be the only supplier. The server ID is `ldap8003`. It will supply updates to the LDAP server on `WTSC81.ITSO.IBM.COM:13389`.
 - c. Consumer(s): The LDAP server on `WTSC81.ITSO.IBM.COM:13389` will be the only consumer. The server ID is `ldap8103`. It will consume updates from the LDAP server on `WTSC80.ITSO.IBM.COM:13389`.
 - d. Read-write server(s): The LDAP server on `WTSC80.ITSO.IBM.COM:13389` with ID `ldap8003` will be the only read-write server.
 - e. Read-only server(s): The LDAP server on `WTSC81.ITSO.IBM.COM:13389` with ID `ldap8103` will be the only read-only server.
2. The following configuration changes are needed for the master and the replica server for replication to work correctly.

- a. Modify the server-ID of master server to human-readable format. In this example, we have used server-ID `ldap8003`, which is the instance name.

- i. Create an ldif file on the master server, name `srvidmaster.ldif`, and paste in the following content:

```
dn: cn=configuration
changetype: modify
replace: ibm-slapdserverid
ibm-slapdserverid: ldap8003
```

- ii. Use `ldapmodify` command to load the `srvidmaster.ldif` file:

```
ldapmodify -h masterldaphost -p port -D adminDN -w passwd -k -f
srvidmaster.ldif
```

For example:

```
ldapmodify -h WTSC80.ITSO.IBM.COM -p 13389 -D "cn=LDAP Admin, o=ibm,
c=us" -w sec001ret -k -f srvidmaster.ldif
```

You should see the following message:

```
modifying entry cn=configuration
```

- b. Modify the server-ID of replica to human-readable format. In this example, we have used server-ID `ldap8103`, which is the instance name.

- i. Create an ldif file on the replica server, name it `srvidreplica.ldif`, and paste in the following content:

```
dn: cn=configuration
changetype: modify
replace: ibm-slapdserverid
ibm-slapdserverid: ldap8103
```

- ii. Use `ldapmodify` command to load the `srvidreplica.ldif` file:

```
ldapmodify -h replicaldaphost -p port -D adminDN -w passwd -k -f
srvidreplica.ldif
```

For example:

```
ldapmodify -h WTSC81.ITS0.IBM.COM -p 13389 -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -k -f srvidreplica.ldif
```

You should see the following message:

```
modifying entry cn=configuration
```

- c. Define replica side credentials entry, the user name, and password that will be used to authenticate bind request from master server:

- i. Create an ldif file on the replica server, name it replicacred.ldif, and paste in the following content:

```
dn: cn=Master server, cn=configuration
cn: master server
ibm-slapdMasterDN: cn=bindtoconsumer
ibm-slapdMasterPW: iamsupplier
ibm-slapdMasterReferral: ldap://WTSC80.ITS0.IBM.COM:13389
objectclass: ibm-slapdReplication
```

- ii. Use **ldapmodify** command to load the replicacred.ldif file:

```
ldapmodify -h replicaldaphost -p port -D adminDN -w passwd -k -f replicacred.ldif
```

Example:

```
ldapmodify -h WTSC81.ITS0.IBM.COM -p 13389 -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -k -f replicacred.ldif
```

You should see the following message:

```
adding new entry cn=Master server, cn=configuration
```

- d. Restart the master and replica servers to put into effect the changes detailed above.

Note: Be careful with server IDs. If a server ID is being used in another topology, changing the server ID can lead to unwanted consequences. Generally, use the instance name as the server ID of the LDAP server.

3. Create a masterreplica.ldif file. Simply copy each of these entries to masterreplica.ldif with the necessary changes in the subtree, server IDs, host names, and ports:

```
dn: o=ibm, c=us
changetype: add
objectclass: top
objectclass: organization
objectclass: ibm-replicationContext
o: ibm
```

```
dn: ibm-replicaGroup=default, o=ibm, c=us
changetype: add
objectclass: top
objectclass: ibm-replicaGroup
ibm-replicaGroup: default
```

```
dn: ibm-replicaServerId=ldap8003,ibm-replicaGroup=default, o=ibm, c=us
changetype: add
objectclass: top
objectclass: ibm-replicaSubentry
```

```
ibm-replicaServerId: ldap8003
ibm-replicationServerIsMaster: true
cn: Master
description: Master server of the topology.
```

```
dn: ibm-replicaServerId=ldap8103,ibm-replicaGroup=default, o=ibm, c=us
changetype: add
objectclass: top
objectclass: ibm-replicaSubentry
ibm-replicaServerId: ldap8103
ibm-replicationServerIsMaster: false
cn: Replica
description: Replica server of the topology.
```

```
dn: cn=ReplicaBindCredentials, o=ibm, c=us
changetype: add
objectclass: ibm-replicationCredentialsSimple
cn: ReplicaBindCredentials
replicaBindDN: cn=bindtoconsumer
replicaCredentials: iamsupplier
description: Bind Credentials on master to bind to replica.
```

```
dn: cn=Replica,
ibm-replicaServerId=ldap8003,ibm-replicaGroup=default,o=ibm,c=us
changetype: add
objectclass: top
objectclass: ibm-replicationAgreement
cn: Replica
ibm-replicaConsumerId: ldap8103
ibm-replicaUrl: ldap://WTSC81.ITS0.IBM.COM:13389
ibm-replicaCredentialsDN: cn=ReplicaBindCredentials, o=ibm, c=us
description: Replication agreement from master to replica.
```

4. The LDIF will be different if the replication context exists. Rather than:

```
dn: o=ibm,c=us
changetype: add
objectclass: top
objectclass: organization
objectclass: ibm-replicationContext
o: ibm
```

It should be:

```
dn: o=ibm,c=us
changetype: modify
add: objectclass
objectclass: ibm-replicationContext
```

5. Load the `masterreplica.ldif` file on the master server. Use `ldapmodify` command to load the `masterreplica.ldif` file on the master server. The option `-k` sends an administrative control to server, and `-L` prevents replication.

```
ldapmodify -h masterldaphost -p port -D adminDN -w passwd -k -L -f
masterreplica.ldif
```

For example:

```
ldapmodify -h WTSC80.ITS0.IBM.COM -p 13389 -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -k -L -f masterreplica.ldif
```

You should see output similar to the following:

```
adding new entry o=ibm, c=us
```

```
adding new entry ibm-replicaGroup=default, o=ibm, c=us
```

```
adding new entry ibm-replicaServerId=ldap8003,ibm-replicaGroup=default, o=ibm, c=us
```

```
adding new entry ibm-replicaServerId=ldap8103,ibm-replicaGroup=default, o=ibm, c=us
```

```
adding new entry cn=ReplicaBindCredentials, o=ibm, c=us
```

```
adding new entry cn=Replica, ibm-replicaServerId=ldap8003, ibm-replicaGroup=default, o=ibm,c=us
```

6. Add replication topology to replica server using extended operation. One can instead export the ldif and reimport the same ldif on replica server using the **ds2ldif** and **ldif2ds** commands. Use the **ldapexop** command to do so:

```
ldapexop -h masterldaphost -p port -D adminDN -w passwd -op repltopology -rc o=ibm,c=us
```

For example:

```
ldapexop -h WTSC80.ITS0.IBM.COM -p 13389 -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -op repltopology -rc "o=ibm,c=us"
```

You should see output similar to the following:

```
repl_topology_extended_op: Success  
repl_topology_extended_op: additional info: R010778 1 servers synchronized  
successfully out of 1 attempts (repl_topology_req)
```

The successful execution of the command listed above indicates replication has been configured.

7. Restart replica server to start replication:

```
P LDAP8001  
S LDAP8001
```

8. Restart master server to start replication:

```
P LDAP8101  
S LDAP8101
```

11.2.4 Peer to peer replication topology configuration in advanced replication

Configuring a peer to peer scenario involves the following steps:

1. Define the configuration parameters:
 - a. Replication context: This entry will be the starting point of the replication, and it can be located anywhere in the tree, not necessary to be suffix like basic replication. In the following configuration example it is considered as `o=ibm,c=us`.

- b. Supplier(s): LDAP server on WTSC80.ITS0.IBM.COM:13389 with server ID Peer1 will supply updates to the LDAP server with server ID Peer2 on WTSC81.ITS0.IBM.COM:13389. LDAP server on WTSC81.ITS0.IBM.COM:13389 with server ID Peer2 will supply updates to the LDAP server with server ID Peer1 on WTSC80.ITS0.IBM.COM:13389.
- c. Consumer(s): LDAP server with server ID Peer2 on WTSC81.ITS0.IBM.COM:13389 will consume updates from LDAP server with server ID Peer1 on WTSC80.ITS0.IBM.COM:13389. LDAP server with Server ID Peer1 on WTSC80.ITS0.IBM.COM:13389 will consume updates from LDAP server with server ID Peer2 on WTSC81.ITS0.IBM.COM:13389.
- d. Read-write server(s): LDAP servers peer1 and peer2 on WTSC80.ITS0.IBM.COM:13389 and WTSC81.ITS0.IBM.COM:13389 will be read-write servers.
- e. Read-only server(s): There are no read-only servers in this topology.

2. Below given configuration changes are needed for the peer1 and the peer2 server for replication to work correctly.

- a. Modify the server-ID of peer1 server to human-readable format. In this example, we have used server-ID ldap8003, which is the instance name.

- i. Create an ldif file on the master server, name it srvidpeer1.ldif, and copy the following contents into it:

```
dn: cn=configuration
changetype: modify
replace: ibm-slapsdserverid
ibm-slapsdserverid: ldap8003
```

- ii. Use **ldapmodify** command to load the srvidpeer1.ldif file:

```
ldapmodify -h peer1ldaphost -p port -D adminDN -w passwd -k -f
srvidpeer1.ldif
```

For example:

```
ldapmodify -h WTSC80.ITS0.IBM.COM -p 13389 -D "cn=LDAP Admin, o=ibm,
c=us" -w sec001ret -k -f srvidpeer1.ldif
```

You should see the following message:

```
modifying entry cn=configuration
```

- b. Modify the server-ID of peer2 to human-readable format. In this example, we have used server-ID ldap8103, which is the instance name.

- i. Create an ldif file on the peer2 server, name it srvidpeer2.ldif, and copy the following contents into it:

```
dn: cn=configuration
changetype: modify
replace: ibm-slapsdserverid
ibm-slapsdserverid: ldap8103
```

- ii. Use **ldapmodify** command to load the srvidpeer2.ldif file:

```
ldapmodify -h replicaldaphost -p port -D adminDN -w passwd -k -f
srvidpeer2.ldif
```

For example:

```
ldapmodify -h WTSC81.ITS0.IBM.COM -p 13389 -D "cn=LDAP Admin, o=ibm,
c=us" -w sec001ret -k -f srvidpeer2.ldif
```

You should see the following message:

```
modifying entry cn=configuration
```

- c. Define replica side credentials entry, the user name, and password that will be used to authenticate bind request from master server:

- i. Create an ldif file on the peer1 server, name it cred.ldif, and copy the following contents into it:

```
dn: cn=Master server, cn=configuration
cn: master server
ibm-slapdMasterDN: cn=bindtoconsumer
ibm-slapdMasterPW: iamsupplier
ibm-slapdMasterReferral: ldap://WTSC80.ITS0.IBM.COM:13389
objectclass: ibm-slapdReplication
```

- ii. Use **ldapmodify** command to load the cred.ldif file on the peer1 server:

```
ldapmodify -h peer1ldaphost -p port -D adminDN -w passwd -k -f cred.ldif
```

For example:

```
ldapmodify -h WTSC80.ITS0.IBM.COM -p 13389 -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -k -f cred.ldif
```

You should see the following message:

```
adding new entry cn=Master server, cn=configuration
```

- iii. Use **ldapmodify** command to load the cred.ldif file on the peer2 server:

```
ldapmodify -h peer2ldaphost -p port -D adminDN -w passwd -k -f cred.ldif
```

For example:

```
ldapmodify -h WTSC81.ITS0.IBM.COM -p 13389 -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -k -f cred.ldif
```

You should see the following message:

```
adding new entry cn=Master server, cn=configuration
```

- d. Restart the peer1 and peer2 servers to put into effect the above changes.

Note: Be careful with server IDs. If a server ID is being used in another topology, changing the server ID can lead to unwanted consequences. Generally, use the instance name as a server ID of the LDAP server.

3. Create a peer2peer.ldif file. Simply copy each of these entries to peer2peer.ldif with the necessary changes in the subtree, server IDs, host names, and ports:

```
dn: o=ibm, c=us
changetype: add
objectclass: top
objectclass: organization
objectclass: ibm-replicationContext
o: ibm
```

```
dn: ibm-replicaGroup=default, o=ibm, c=us
changetype: add
objectclass: top
objectclass: ibm-replicaGroup
ibm-replicaGroup: default
```

```
dn: ibm-replicaServerId=ldap8003,ibm-replicaGroup=default, o=ibm, c=us
changetype: add
objectclass: top
objectclass: ibm-replicaSubentry
ibm-replicaServerId: ldap8003
ibm-replicationServerIsMaster: true
cn: Peer1
description: Subentry for Peer1.
```

```
dn: ibm-replicaServerId=ldap8103,ibm-replicaGroup=default, o=ibm, c=us
changetype: add
objectclass: top
objectclass: ibm-replicaSubentry
ibm-replicaServerId: ldap8103
ibm-replicationServerIsMaster: true
cn: Peer2
description: Subentry for Peer2.
```

```
dn: cn=ReplicaBindCredentials, o=ibm, c=us
changetype: add
objectclass: ibm-replicationCredentialsSimple
cn: ReplicaBindCredentials
replicaBindDN: cn=bindtoconsumer
replicaCredentials: iamsupplier
description: Bind Credentials on peer1 and peer2 to bind to each other.
```

```
dn: cn=Peer2, ibm-replicaServerId=ldap8003,ibm-replicaGroup=default,o=ibm,c=us
changetype: add
objectclass: top
objectclass: ibm-replicationAgreement
cn: Peer2
ibm-replicaConsumerId: ldap8103
ibm-replicaUrl: ldap://WTSC81.ITS0.IBM.COM:13389
ibm-replicaCredentialsDN: cn=ReplicaBindCredentials, o=ibm, c=us
description: Replication agreement from peer1 to peer2.
```

```
dn: cn=Peer1, ibm-replicaServerId=ldap8103,ibm-replicaGroup=default,o=ibm,c=us
changetype: add
objectclass: top
objectclass: ibm-replicationAgreement
cn: Peer1
ibm-replicaConsumerId: ldap8003
ibm-replicaUrl: ldap://WTSC80.ITS0.IBM.COM:13389
ibm-replicaCredentialsDN: cn=ReplicaBindCredentials, o=ibm, c=us
description: Replication agreement from peer2 to peer1.
```

4. The LDIF will be different if the replication context exists. Rather than:

```
dn: o=ibm,c=us
changetype: add
objectclass: top
objectclass: organization
objectclass: ibm-replicationContext
o: ibm
```

It should be:

```
dn: o=ibm,c=us
changetype: modify
add: objectclass
objectclass: ibm-replicationContext
```

5. Load the peer2peer.ldif file on the peer1 server Use the **ldapmodify** command to load the peer2peer.ldif file on the peer1 server. The option **-k** sends an administrative control to server, and **-L** prevents replication.

```
ldapmodify -h peer1ldaphost -p port -D adminDN -w passwd -k -L -f
masterreplica.ldif
```

For example:

```
ldapmodify -h WTSC80.ITS0.IBM.COM -p 13389 -D "cn=LDAP Admin, o=ibm, c=us" -w
sec001ret -k -L -f peer2peer.ldif
```

You should see output similar to the following:

```
adding new entry o=ibm, c=us
```

```
adding new entry ibm-replicaGroup=default, o=ibm, c=us
```

```
adding new entry ibm-replicaServerId=ldap8003,ibm-replicaGroup=default, o=ibm,
c=us
```

```
adding new entry ibm-replicaServerId=ldap8103,ibm-replicaGroup=default, o=ibm,
c=us
```

```
adding new entry cn=ReplicaBindCredentials, o=ibm, c=us
```

```
adding new entry cn=Peer2,
ibm-replicaServerId=ldap8003,ibm-replicaGroup=default,o=ibm,c=us
```

```
adding new entry cn=Peer1,
ibm-replicaServerId=ldap8103,ibm-replicaGroup=default,o=ibm,c=us
```

6. Add the replication topology to the peer2 server using extended operation. You can export the data in an ldif file and reimport the same ldif file on peer2 server using the **ds2ldif** and **ldif2ds** commands. Use the **ldapexop** command to do so:

```
ldapexop -h peer1ldaphost -p port -D adminDN -w passwd -op repltopology -rc
o=ibm,c=us
```

For example:

```
ldapexop -h WTSC80.ITS0.IBM.COM -p 13389 -D "cn=LDAP Admin, o=ibm, c=us" -w
sec001ret -op repltopology -rc "o=ibm,c=us"
```

You should see output similar to the following:

```
repl_topology_extended_op: Success
repl_topology_extended_op: additional info: R010778 1 servers synchronized
successfully out of 1 attempts (repl_topology_req)
```

If the above command executes successfully, it means replication has been configured.

7. Restart the peer1 server to start replication:

```
P LDAP8001
```

S LDAP8001

8. Restart the peer2 server to start replication:

P LDAP8101

S LDAP8101



Using LDAP and HCD

This chapter provides a simple configuration to enable LDAP support for HCD, and basic examples to demonstrate its use.

For more information, refer to the *Hardware Configuration Definition User's Guide*, SC33-7988 and *I/O Configuration Using z/OS HCD and HCM*, SG24-7804.

12.1 Hardware Configuration Definition (HCD) and LDAP

HCD LDAP plug-in, along with IBM Tivoli Directory Server for z/OS (TDS) and the RACF back end SDBM, can be used to access and update existing IODFs through the standardized Lightweight Directory Access Protocol (LDAP), which is based on TCP/IP.

The HCD LDAP plug-in is optional and there are no limits in HCD functionality if the plug-in is not used.

LDAP is a protocol that makes directory information available (like the yellow pages). New entries can be added, existing entries altered or deleted, and you can search for matching entries using wildcards.

The HCD LDAP plug-in is plugged into the IBM Tivoli Directory Server and configured using the IBM Tivoli Directory Server for z/OS configuration file (usually called `ds.conf`).

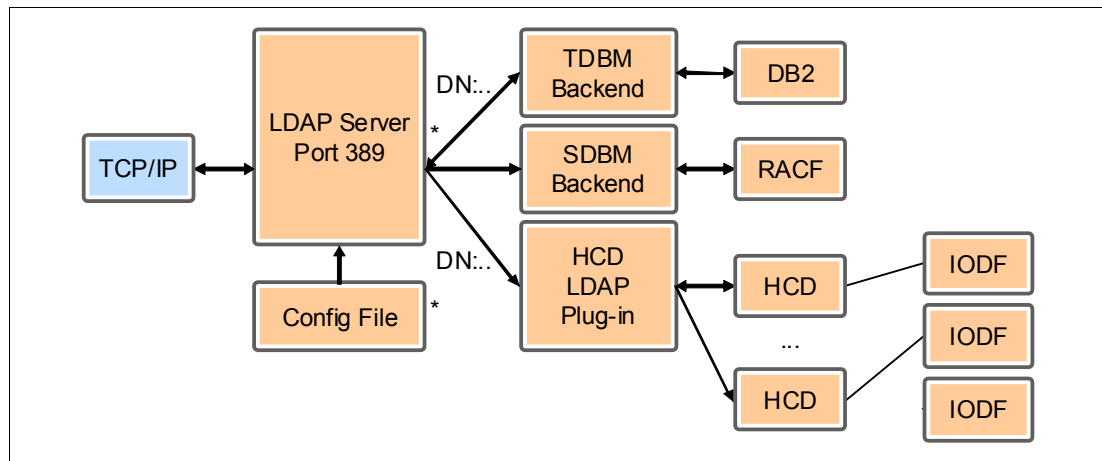


Figure 12-1 HCD LDAP plug-in structure

Similar to the RACF plug-in SDBM, the main function of the HCD LDAP plug-in is to mediate between the IBM Tivoli Directory Server and an external component, in this case HCD.

HCD maintains control of the IODFs while update requests are validated and processed. The results are stored in the appropriate IODF. Even though the IBM Tivoli Directory Server updates the IODF, HCD processes the updates, ensuring the integrity of the IODFs.

Because of this, the HCD portion of the LDAP Directory Information Tree (DIT) must reflect the data structure of HCD exactly. There are strict rules invoked when requesting an update of IODF data through the IBM Tivoli Directory Server.

The HCD LDAP plug-in uses only RACF access rights of the user IDs that services are performed for to determine whether this is a valid request or not. This assumes that the plug-in runs under a user ID that is entitled to switch to the user ID of the respective bind request. The plug-in takes its user ID as that of the IBM Tivoli Directory Server. Therefore, the HCD LDAP plug-in can only be plugged into the IBM Tivoli Directory Server, and the IBM Tivoli Directory Server must run as a started task under a user ID that is permitted to switch to another user ID. To perform the switching, the HCD LDAP plug-in uses the `pthread_security_np()` service (thread-level security model).

You will need to choose from two options for the security level of IBM Tivoli Directory Server:

▶ UNIX level security

The IBM Tivoli Directory Server must run under a superuser who is automatically entitled to assume the identity of any other user.

▶ z/OS UNIX level security

The right to switch user IDs must be explicitly granted to all, even to the superuser. The z/OS UNIX level security should be used because it is more secure than the UNIX level. Be aware that this is a global decision that can affect all the servers on your system. Steps required are listed for both options, and warnings are issued whenever a step will affect your system configuration.

The setup should be divided into three parts:

1. Set up the IBM Tivoli Directory Server for z/OS (TDS) so that it can run with the HCD LDAP plug-in.
2. Set up the HCD LDAP plug-in as plug-in to the IBM Tivoli Directory Server (including security definitions, APF authority, and so on).
3. Add the HCD schema (`schema.hcd.ldif` located in `/usr/lpp/hcd/etc` directory) into IBM Tivoli Directory Server.

These are all performed outside of HCD, so refer to the z/OS HCD User's Guide (Version 1 Release 11) Chapter 14, page 351 for setup information.

There is a one-to-one mapping between the IODF data structure and the LDAP directory information tree (DIT) that makes the HCD IODF information able to be accessed through the LDAP protocol and makes IODF updates possible.

There are two steps you must always perform to request a service from the HCD LDAP plug-in:

1. You must authenticate (bind) yourself to the RACF back end SDBM.
2. You must access an IODF.

The HCD LDAP plug-in functions as follows:

- ▶ HCD LDAP plug-in does NOT participate in extended group membership searching a client request.
- ▶ It is possible to run several back ends on one IBM Tivoli Directory Server.
- ▶ The root of a subtree or back end is denoted with a suffix in the configuration file. There is only one suffix specified per HCD LDAP plug-in, and it must be unique if you are running multiple HCD LDAP plug-ins.
- ▶ It does NOT support Access Control Lists (ACLs) that are used to protect information stored in an LDAP directory from unauthorized access because the access control is performed by RACF.
- ▶ It does NOT support LDAP request types such as Bind, ModifyDN and ModifyRDN, Compare, Abandon, and Extended Request. Requests will receive the return code `Unwilling to Perform`.
- ▶ It does support the following LDAP request types but does impose restrictions due to the fact that the HCD portion of the DIT is rigidly controlled:
 - Add
 - Delete
 - Modify
 - Search

- ▶ HCD LDAP plug-in does NOT support multi-server or replication.

To initiate, extend, and close a transaction using the HCD LDAP plug-in, you must have set up and run the IBM Tivoli Directory Server and the back end, and you must provide an LDAP V3 client program with the appropriate controls of the HCD LDAP plug-in. You must select the version of the LDAP client API function that allows the specification of server controls. See the *IBM Tivoli Directory Server Client Programming for z/OS V1R12.0*, SA23-2214-04 for more information about the functions and the parameters to be passed for requests.

12.2 Securing IBM Tivoli Directory Server for z/OS HCD

There are two options for securing IBM Tivoli Directory Server with z/OS:

- ▶ UNIX level security
- ▶ z/OS UNIX level security

With UNIX level security, IBM Tivoli Directory Server for z/OS must run under the superuser (uid=0). The superuser on this security level has total authority over the system. In particular, the user is automatically entitled to assume the identity of any other user.

With z/OS UNIX level security, the right to switch user IDS must be explicitly granted even to the superuser. This is the preferred level of security.

If your environment is not currently set up for z/OS UNIX level security, be aware that enabling this level of security is a global change and might impact other areas in your z/OS environment.

The examples in this chapter assume that you are using z/OS UNIX level security.

12.3 Configuring HCD and LDAP

This section shows the configuration options needed in HCD, and the associated changes needed in the configuration file for IBM Tivoli Directory Server.

The following prerequisites must be met to use HCD with IBM Tivoli Directory Server:

- ▶ IBM Tivoli Directory Server must be set up to run as a started task.
- ▶ Establish a separate user ID that runs the IBM Tivoli Directory Server for z/OS.
- ▶ Set up the IBM Tivoli Directory Server for z/OS for running in single-server mode without replication.
- ▶ Configure the IBM Tivoli Directory Server for z/OS with the SDBM back end. The HCD LDAP plug-in requires the IBM Tivoli Directory Server for z/OS to run with this RACF back end. Therefore, SDBM must be included in the configuration file and all prerequisites for SDBM must be fulfilled.

In addition, to use z/OS UNIX level security:

- ▶ The RACF FACILITY profile BPX.SERVER must be defined. For more information, refer to *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 and *z/OS UNIX System Services Planning*, GA22-7800.
- ▶ Certain libraries have to be defined to RACF program control. Work with your RACF administrator because changes to the RACF class PROGRAM can cause problems in your z/OS environment if not done correctly.

12.3.1 Setting up the IBM Tivoli Directory Server for z/OS

Divide the setup process into three parts:

- ▶ Set up the IBM Tivoli Directory Server for z/OS so that it is able to run with the HCD LDAP plug-in.
- ▶ Set up the HCD LDAP plug-in.
- ▶ Integrate the HCD schema into the IBM Tivoli Directory Server for z/OS.

Setting up IBM Tivoli Directory Server with SDBM back end

To set up IBM Tivoli Directory Server, perform the following steps:

1. Copy the following sample files from `/usr/lpp/ldap/etc` to a new location. You can use `/etc`. However, because we are implementing multiple instances of IBM Tivoli Directory Server on a single system, we chose to create a home directory under `/u` for each instance.
 - `ds.profile`
 - `ds.slapped.profile`
 - `ds.racf.profile`
 - `ds.db2.profile` (not changed in this implementation, but `ds.profile` has a link to this file)
2. Modify `ds.profile` with the following changes:
 - `ADMINDN = "cn=root"`
 - `ADMINPW = root`
 - `SDBM_SUFFIX = "o=ibm,c=us"`
 - `PROG_SUFFIX = HC`
 - `LDAPUSRID = LDAPHCD`
 - `LDAPUSRGRP = LDAPGRP`
 - `SCHEMAPATH = /u/ldaphcd/schema`
 - `OUTPUT_DATASET = LDAPHCD.CNFOUT`
 - `OUTPUT_DATASET_VOLUME = BH8ST4`
 - Added jobcard statements for `APF_JOB CARD`, `PRGCTRL_JOB CARD`, `DB2_JOB CARD`, and `RACF_JOB CARD`
 - `SLAPD_PROFILE = /u/ldaphcd/ds.slapped.profile`
 - `DB2_PROFILE = /u/ldaphcd/ds.db2.profile`
 - `RACF_PROFILE = /u/ldaphcd/ds.racf.profile`
3. Modify `ds.slapped.profile` with the following changes:
 - `ARMNAME = LDAPHCD`
 - `LISTEN = ldap://:5389` (this port must be unique for each ldap running on a single system)
 - `LOGFILE = /u/ldaphcd/logs/gldlog.output`

Note: The `/logs` subdirectory will not be created by IBM Tivoli Directory Server, and must exist prior to starting IBM Tivoli Directory Server. Also verify that the LDAP server's user ID has write access to this entire directory.

4. Modify `ds.racf.profile` with the following changes:
 - `LDAPGID = 22` (group ID must be a unique decimal number. Do not use gid 0.)
 - `LDAPUID = 330` (user ID must be unique decimal number. Do not use uid 0.)
5. Prior to running `dsconfig`, the following variables must be exported on the z/OS UNIX System Services session where `dsconfig` will be run:
 - `export STEPLIB=SYS1.SIEALNKE:$STEPLIB`

- export PATH=/usr/lpp/ldap/sbin:\$PATH
 - export NLSPATH=/usr/lpp/ldap/lib/nls/msg/%L/%N:\$NLSPATH
 - export LANG=En_US.IBM-1047
6. Run **dsconfig** with the `ds.profile` file that was customized:
- ```
dsconfig -i /u/ldaphcd/ds.profile
```
- You should see the following messages:
- ```
100623 14:20:06.502052 GLD2002I Directory Server configuration utility has
started.
100623 14:20:06.873252 GLD2003I Directory Server configuration utility has
ended.
```
7. Access the PDS in TSO that was created by **dsconfig**. You should find the following members were created:
- RACF: Creates the `ldaphcd` user and `ldapgrp` group, and grants authority to RACF-protected resources
 - PROGxx: APF-authorizes certain datasets for `ldap`
 - APF: Issues the console command `SET PROG=XX`, which points to the `progxx` member that was created by **dsconfig**
 - LDAPHCD: The proc to run the `ldap` started task.
 - PRGCNTRL: Sets program control for related libraries
 - DSCONFIG: Configuration file for `ldap`
 - DSENVVAR: Environment variables that must be used for running `ldap`
8. Copy `PROGxx` to a system `parmlib`.
9. Submit job `APF` to authorize the datasets, created by **dsconfig**, listed in `PROGxx`.

Important: The datasets *must* be added to a the system APF list to make authorizations persistent across IPLs.

10. Submit job **RACF** to grant the RACF permissions necessary for the server to run. If using a security product other than RACF, make the necessary modifications to this job prior to submitting.
11. Copy the LDAP started task JCL (in our example, `LDAPHCD`) to a system `proclib`.

Starting and verifying IBM Tivoli Directory Server operation

To start IBM Tivoli Directory Server, issue a console start command for the started task JCL that was created by **dsconfig** and you copied to a system `proclib`. Refer to the comments within the started task proc for parameters that can be specified when starting IBM Tivoli Directory Server. To start IBM Tivoli Directory Server with no parameters, enter **S *LDAPHCD*** on the console, replacing *LDAPHCD* with the name of your server in `proclib`.

Verify a successful start by watching for the following messages:

```
GLD1004I LDAP server is ready for requests.
GLD1059I Listening for requests on 9.12.4.47 port 5389.
GLD1059I Listening for requests on 9.12.4.61 port 5389.
GLD1059I Listening for requests on 127.0.0.1 port 5389.
```

To verify IBM Tivoli Directory Server functionality, enter the following command from z/OS UNIX System Services:

```
ldapsearch -h 127.0.0.1 -p 5389 -s base -b "" "objectclass=**"
```

Load the IBM-supplied schema

z/OS IBM Tivoli Directory Server stores schema as an entry in the database. The distinguished name of the schema entry is `cn=schema`. Use `ldapsearch` with `cn=schema` base to list all schema entries in the running IBM Tivoli Directory Server instance:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s base -b "cn=schema" objectclass=*
```

Replace `ldaphost` with your host name, `port` with the correct portnumber, and `adminDN` and `passwd` with the administrator distinguished name and password.

For example:

```
ldapsearch -D cn=root -w root -p 5389 -s base -b "cn=schema" objectclass=*
```

The value of the `schemaPath` option in the IBM Tivoli Directory Server configuration file defines the location where IBM Tivoli Directory Server stores schema entry. The z/OS IBM Tivoli Directory Server instance owner needs to have read and write permission to the schema location. The default value is `/var/ldap/schema`. If multiple instances need to be configured on the one system, do not to use the default location for all instances. Instead, use separate locations for all instances.

On first start-up IBM Tivoli Directory Server creates an initial default schema that is sufficient for usage of the GDBM, CDBM, and SDBM (w/o custom fields), but needs to be updated for LDBM, TDBM, SDBM with RACF custom fields, and CDBM with user-defined entries.

z/OS IBM Tivoli Directory Server is shipped with predefined schema files, `schema.IBM.ldif` and `schema.user.ldif`. Use the `ldapmodify` command to load the schema files in running IBM Tivoli Directory Server instance. The commands to load the `schema.user.ldif` and `schema.IBM.ldif` schema files are:

```
ldapmodify -h ldaphost -p ldapport -D adminDN -w passwd -f schemafile
```

Replace `ldaphost` with your host name, `ldapport` with the correct portnumber, and `adminDN` and `password` with the administrator distinguished name and password.

For example:

```
ldapmodify -D cn=root -w root -p 5389 -f /usr/lpp/ldap/etc/schema.user.ldif 2>&1 | tee user.out
```

```
ldapmodify -D cn=root -w root -p 5389 -f /usr/lpp/ldap/etc/schema.IBM.ldif 2>&1 | tee IBM.out
```

For both these commands, you should see the following message:

```
modifying entry cn=schema
```

12.3.2 Setting up the HCD LDAP plug-in

To set up the HCD LDAP plug-in, perform the following steps.

1. Authorize the HCD LDAP plug-in to act on behalf of other user IDs. z/OS UNIX level requires that all users, including the superuser, be explicitly authorized to switch user IDs.
 - a. Define a surrogate profile for the prospective client by issuing the following RACF commands. The second command updates the in-storage copy of the SURROGAT profiles.

```
RDEFINE SURROGAT BPX.SRV.userID UACC(NONE)
```

```
SETRPTS RACLIST(SURROGAT) REFRESH
```

- b. Authorize the user ID of the IBM Tivoli Directory Server for z/OS for this profile using the following commands. The second command updates the in-storage copy of the SURROGAT profiles.

```
PERMIT BPX.SRV.userID CLASS(SURROGAT) ID(LDAPHCD) ACCESS(READ)
SETRPTS RACLIST(SURROGAT) REFRESH
```

These example commands are based on the following assumptions (which might not hold for your system):

- The RACF class **SURROGAT** has been activated.
- There is no profile in that class with the name `BPX.SRV.userID`, where *userID* is the user ID of the prospective client.
- The user ID of the IBM Tivoli Directory Server for z/OS on our system is LDAPHCD.

2. Define libraries to program control. The libraries containing the following load modules must be defined to RACF program control:

- HCD LDAP plug-in (typically SYS1.LINKLIB on SYSRES)
- HCD (typically SYS1.LINKLIB on SYSRES)
- UIMs (typically SYS1.NUCLEUS on SYSRES)
- C++ RTL (typically CEE.SCEERUN on SYSRES)
- IBM Tivoli Directory Server (typically SYS1.SIEALNKE and SYS1.LPALIB on SYSRES)

To define these libraries to program control, issue the following RACF commands:

```
RDEFINE PROGRAM ** UACC(READ)
RALTER PROGRAM ** UACC(READ) ADDMEM('SYS1.LINKLIB'/'*****'/NOPADCHK)
RALTER PROGRAM ** UACC(READ) ADDMEM('SYS1.NUCLEUS'/'*****'/NOPADCHK)
RALTER PROGRAM ** UACC(READ) ADDMEM('CEE.SCEERUN'/'*****'/NOPADCHK)
RALTER PROGRAM ** UACC(READ) ADDMEM('SYS1.SIEALNKE'/'*****'/NOPADCHK)
RALTER PROGRAM ** UACC(READ) ADDMEM('SYS1.LPALIB'/'*****'/NOPADCHK)
SETRPTS WHEN(PROGRAM) REFRESH
```

The first command defines a profile named ****** to the class PROGRAM. The other commands, except the last, define the libraries containing the load modules to program control. The last command refreshes the in-storage copy of the PROGRAM profiles. The example commands are based on the following assumptions (which might not hold for your system):

- The RACF class PROGRAM has been activated.
- GENERIC is enabled for the RACF class PROGRAM.
- There is no profile in that class with the name ******.
- The load modules needed reside in their typical libraries as listed above.

3. APF authorize libraries. The libraries containing the following load modules must be APF authorized:

- HCD LDAP plug-in (typically SYS1.LINKLIB on SYSRES)
- C++ RTL (typically CEE.SCEERUN on SYSRES)
- IBM Tivoli Directory Server (typically SYS1.SIAELNKE and SYS1.LPALIB on SYSRES)

4. Tailor the started task procedure. This includes:

- The HCD instances that have been started by the HCD LDAP plug-in have the same region size as the IBM Tivoli Directory Server for z/OS started task. Therefore, you might need to adjust the region size of the IBM Tivoli Directory Server for z/OS started task according to the region size suitable for the HCD instances.

- Ensure that the IBM Tivoli Directory Server for z/OS and the HCD LDAP plug-in are able to find the load modules that were defined to RACF program control by using the z/OS search order. If the libraries containing these load modules are not searched by z/OS on your system, you must insert a STEPLIB DD, which contains the missing libraries, into the started task procedure.

5. Tailor the IBM Tivoli Directory Server for z/OS configuration file.

You must include the definition of the HCD LDAP plug-in in the IBM Tivoli Directory Server configuration file. A sample of how to define the HCD LDAP plug-in as IBM Tivoli Directory Server for z/OS plug-in in the server configuration file is delivered with HCD and is installed in `/usr/lpp/hcd/examples/ds.conf`. For this purpose you must add into the GLOBAL section of the configuration file the following option to define the HCD LDAP plug-in as a plug-in:

```
plugin clientOperation CBDMLPLG hcd_plginit "<parameters>"
```

See Figure 12-2.

```

EDIT      LDAPHCD.CNFOUT(DSCONFIG) - 01.01          Columns 00001 00072
Command ==>>>                                     Scroll ==>>> CSR
001146 #-----
001147 # HCD Plug-in
001148 #-----
001149 plugin clientOperation CBDMLPLG hcd_plginit
001150 "suffix          cn=HCD
001151 MinHcdInstances  1
001152 MaxHcdInstances  3
001153 AllowSwitchTime  30
001154 ForceSwitchTime  600
001155 TransactionRollbackTime 3600
001156 Trace           off
001157 Profile         off
001158 TraceDsnSuffix  HCD.TRACE
001159 ProfileDsnSuffix HCD.PROFILE
001160 TransformAttributeValues off"
001161
001162 # =====

```

Figure 12-2 HCD Plug-in added to GLOBAL section of configuration file

6. Run the HCD LDAP plug-in.

To verify that your setup is working, issue an LDAP request against the HCD LDAP plug-in. You can use the LDAP operation utilities to do this. For this purpose, enter a command according to the following template:

```
ldapsearch -h ldaphost -p ldapport -D binddn -w passwd -s base -b
"hcdIodfId=IodfName,suffix" "objectclass=*"

```

This command performs a search on the specified IODF on behalf of the user ID specified by *binddn*. *binddn* must be a DN from within the SDBM name space representing a user ID, and *passwd* the appropriate password. *IodfName* must be the name of an existing IODF data set. *suffix* would be *cn=HCD* if you have kept the default value specified in the sample configuration file.

If the request returns a plausible result, the HCD LDAP plug-in is working correctly:

```
ldapsearch -h 127.0.0.1 -p 5389 -D
"racfid=RGREEN,profiletype=user,cn=racf,o=ibm,c=us" -w "passwd" -s base -b
"hcdIodfId=RGREEN.IODF53.WORK,cn=HCD" "objectclass=*"

```

```
hcdIodfId=RGREEN.IODF53.WORK,cn=HCD
objectClass=hcdIodf
hcdiodfid=RGREEN.IODF53.WORK
hcdiodftype=W
hcdblocksallocated=2216
hcdblocksused=1960
hcdcreationdate=2010-07-13
hcdlastupdatedate=2010-07-13
hcdlastupdatetime=15:55:41
```

12.3.3 Integrating the LDAP schema for HCD

HCD is shipped with a predefined schema file containing schema definitions that the IBM Tivoli Directory Server for z/OS needs to evaluate incoming HCD requests issued through the LDAP interface. You must integrate this file into the IBM Tivoli Directory Server for z/OS after this server has been successfully installed and set up. This integration step should be performed by the person who is responsible for the IBM Tivoli Directory Server for z/OS (usually the system administrator). The name of the HCD schema file is `schema.hcd.ldif` and is located in the `/usr/lpp/hcd/etc` directory. Use the **ldapmodify** command to load the schema:

```
ldapmodify -h ldaphost -p ldapport -D adminDN -w passwd -f
/usr/lpp/hcd/etc/schema.hcd.ldif
```

See *IBM Tivoli Directory Server Client Programming for z/OS V1R12.0*, SA23-2214-04 for more information about **ldapmodify**.

12.4 Using HCD and LDAP

IODF updates, such as adding, deleting, or changing devices, control units, or operating system configs can be performed through LDAP using the standard LDAP commands **ldapadd**, **ldapsearch**, **ldapdelete**, and **ldapmodify**. Complex updates can be done using applications using the client programming interface. The HCD LDAP plug-in supports the concept of a *transaction*, which is a series of individual requests that are executed as a whole. If one of the requests in the transaction fails, then none of the requests are executed. This provides a mechanism for data consistency.

12.4.1 Authentication

To access an IODF through LDAP, you *must* authenticate yourself to the RACF back end. This is done by specifying a bind DN and a password, using the following format:

```
-D "racfid=user_ID,profileType=user,suffix"
```

user_ID is a valid RACF user with authority to use HCD, and *suffix* is the suffix specified in the IBM Tivoli Directory Server configuration file. For example, in the **ldapsearch** command shown earlier to verify operation, this was specified as:

```
-D "racfid=RGREEN,profiletype=user,cn=racf,o=ibm,c=us"
```

Also, the correct password for *user_ID* must be specified with the **-w** option.

A valid IODF dataset must be specified in the following format:

```
-b "hcdIodfId=Iodf_dataset_name,suffix"
```

Iodf_dataset_name is the name of the MVS dataset containing the IODF, and *suffix* is the suffix for the HCD LDAP plug-in specified in the IBM Tivoli Directory Server configuration file. If you took the default, then *suffix* is *cn=HCD*.

In our example **ldapsearch** command, the IODF dataset is specified as:

```
-b "hcdIodfId=RGREEN.IODF53.WORK,cn=HCD"
```

12.4.2 Usage examples

The following are basic examples that show using **ldap** commands to manipulate an IODF. Appendix F. IODF data model in the *Hardware Configuration Definition Users Guide*, SC33-7988 describes the IODF object hierarchy below the HCD plug-in suffix in terms of object class and attribute definitions. Usage examples can be found in that book.

We provide a single example of adding a control unit.

Example: Adding a control unit to an IODF

In this example, a control unit of type 3990 is added to the IODF. The CU number is 0100. The definition is in an MVS dataset.

First create a data set member with the following content:

```
dn:hcdControlUnitNumber=0100,hcdIodfId=RGREEN.IODF53.WORK,cn=HCD
changetype:add
objectclass:hcdControlUnit
hcdControlUnitNumber:0100
hcdUnit:3990
hcdConnPorts:AE.23
```

Then call the LDAP command line utility **ldapadd** to add the control unit, with the **-f** parameter pointing the file that has the definition for the control unit:

```
ldapadd -h 127.0.0.1 -p 5389 -D
"racfid=RGREEN,profiletype=user,cn=racf,o=ibm,c=us" -w "passwd" -f
//'HCD.WORK(ADDCU100)'
```

Expect the following output:

```
adding new entry hcdControlUnitNumber=0100,hcdIodfId=RGREEN.IODF53.WORK,cn=HCD
```




Monitoring

Every component within an IT environment should have an overall performance objective or agreement such as a Service Level Objective (SLO) or a Service Level Agreement (SLA). With IBM Tivoli Directory Server for z/OS, like many applications that are transaction based, it can be difficult to predict its operation and performance on paper. Until it is configured, the data is loaded and the actual client transactions applied, only rules of thumb can be used. The size of the directory objects (fat versus thin), transaction mix (number of clients, adds, modifies, searches, and so on) and the types and quantities of results that must be returned to the clients can vary significantly throughout its service life.

In addition, because the Directory Server is normally a data store/back end for user-facing applications, setting baselines (SLA/Os) and continuously monitoring it to ensure it meets the required performance level can prevent both finger pointing and lost time in problem determination.

This chapter discusses server monitoring, and managing and monitoring advanced replication in z/OS LDAP.

13.1 Server monitoring

An LDAP administrator can retrieve server statistics from the server by executing a search request with a search base of `cn=monitor` and a filter of `objectclass=*`. No other search base and filter are accepted, but any of the possible scope values are accepted.

The z/OS LDAP server presents monitor data in multiple distinguished name entries:

1. `cn=monitor`: This entry returns server-wide statistics.
2. `cn=back endXXXX,cn=monitor`: This entry returns statistics of configured back end whose name is `XXXX`, and `XXXX` is the back end name specified on the database configuration option in the server configuration file. If no back end name is specified on the database configuration option, the LDAP server generates a name.
3. `cn=back endMonitor,cn=monitor`: This entry returns statistics for the back end handling `cn=monitor` searches.
4. `cn=back endSchema,cn=monitor`: This entry returns statistics for the back end managing the schema.
5. `cn=back endRootDSE,cn=monitor`: This entry returns statistics for the back end handling root DSE searches.
6. `cn=operations,cn=monitor`: This entry returns statistics of search patterns, but only if the operations monitor is on (the `operationsMonitorSize` configuration option is not set to zero).

With a valid scope of:

- ▶ **base**: Only the `cn=monitor` entry is returned containing server-wide statistics.
- ▶ **one**: All back end-specific entries are returned and the operations monitor entry is returned (if configured).
- ▶ **sub**: All entries are returned, including the operations monitor entry (if configured).

Note: The statistics reported on the `cn=monitor` subtree search can also be displayed by using the LDAP server `DISPLAY` operator modify command, and operations monitor statistics cannot be displayed by using the `DISPLAY` operator command.

13.1.1 Monitor search with `scope=sub`

The following `ldapsearch` command returns complete LDAP server statistics:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b cn=monitor  
objectclass=*
```

The monitor search returns some of the following attributes of the server:

- ▶ `cn: monitor`
- ▶ `version`: z/OS Version 1 Release 12 IBM Tivoli Directory Server LDAP Server
- ▶ `livethreads`: The number of worker threads being used by the server
- ▶ `maxconnections`: The maximum number of active connections allowed
- ▶ `sysmaxconnections`: The system-defined maximum number of connections
- ▶ `totalconnections`: The total number of connections since the server was started
- ▶ `currentconnections`: The number of active connections

- ▶ maxreachedconnections: The high water mark for concurrent client connections.
- ▶ opsinitiated: The number of operations initiated
- ▶ opscompleted: The number of operations completed
- ▶ abandonsrequested: The number of abandon operations requested
- ▶ abandonscompleted: The number of abandon operations completed
- ▶ addsrequested: The number of add operations requested
- ▶ addscompleted: The number of add operations completed
- ▶ bindsrequested: The number of bind operations requested
- ▶ bindscompleted: The number of bind operations completed
- ▶ comparesrequested: The number of compare operations requested
- ▶ comparescompleted: The number of compare operations completed
- ▶ deletesrequested: The number of delete operations requested
- ▶ deletescompleted: The number of delete operations completed
- ▶ extopsrequested: The number of extended operations requested
- ▶ extopscompleted: The number of extended operations completed
- ▶ modifiesrequested: The number of modify operations requested
- ▶ modifiescompleted: The number of modify operations completed
- ▶ modifydnsrequested: The number of modify DN operations requested
- ▶ modifydnscompleted: The number of modify DN operations completed
- ▶ searchesrequested: The number of search operations requested
- ▶ searchescompleted: The number of search operations completed
- ▶ unbindsrequested: The number of unbind operations requested
- ▶ unbindscompleted: The number of unbind operations completed
- ▶ unknownopsrequested: The number of unknown operations requested
- ▶ unknownopscompleted: The number of unknown operations completed
- ▶ entriessent: The number of search entries sent
- ▶ bytessent: The number of bytes sent
- ▶ searchreferencessent: The number of search references sent
- ▶ currenttime: The current date and time on the server
- ▶ starttime: The date and time the server was started
- ▶ resettime: The date and time the server were last reset
- ▶ resets: The number of times statistics were reset
- ▶ namingcontexts: The suffixes managed by this back end
- ▶ filter_cache_size: The configured maximum size (in entries) of the Filter cache
- ▶ filter_cache_current: The current size (in entries) of the Filter cache
- ▶ filter_cache_hit: The number of lookups that have hit the Filter cache
- ▶ filter_cache_miss: The percent of lookups that have hit the Filter cache
- ▶ filter_cache_percent_hit: The percent of lookups that have hit the Filter cache
- ▶ filter_cache_refresh: The number of times the Filter cache was invalidated

- ▶ filter_cache_refresh_avgsiz: The average number of entries in the Filter cache at invalidation
- ▶ filter_cache_bypass_limit: The configured Filter cache bypass limit
- ▶ dn_cache_size: The configured maximum size (in entries) of the DN cache (dnCacheSize)
- ▶ dn_cache_current: The current size (in entries) of the DN cache
- ▶ dn_cache_hit: The number of lookups that have hit the DN cache
- ▶ dn_cache_miss: The number of lookups that have missed the DN cache
- ▶ dn_cache_percent_hit: The percent of lookups that have hit the DN cache
- ▶ dn_cache_refresh: The number of times the DN cache was invalidated
- ▶ dn_cache_refresh_avgsiz: The average number of entries in the DN cache at invalidation
- ▶ acl_source_cache_size: The configured maximum size (in entries) of the ACL Source cache (aclSourceCacheSize)
- ▶ acl_source_cache_current: The current size (in entries) of the ACL Source cache
- ▶ acl_source_cache_hit: The number of lookups that have hit the ACL Source cache
- ▶ acl_source_cache_miss: The number of lookups that have missed the ACL Source cache
- ▶ acl_source_cache_percent_hit: The percent of lookups that have hit the ACL Source cache
- ▶ acl_source_cache_refresh: The number of times the ACL Source cache was invalidated
- ▶ acl_source_cache_refresh_avgsiz: The average number of entries in the ACL Source cache at invalidation
- ▶ searchStats: The search statistics for search patterns based on the search parameters (search base, scope, filter, and attributes to be returned) and status (success or failure)
- ▶ numtrimmed: The number of search patterns trimmed from the operations monitor
- ▶ entries: The total number of search patterns in the operations monitor entry
- ▶ cachesize: The configured maximum number of search patterns in the operations monitor

Example:

```
ldapsearch -D "cn=LDAP Admin, o=ibm, c=us" -w sec001ret -p 4389 -s sub -b
cn=monitor objectclass=*
```

You should receive output similar to the following:

```
cn=monitor
version=z/OS Version 1 Release 12 IBM Tivoli Directory Server LDAP Server
livethreads=10
maxconnections=65517
sysmaxconnections=65535
totalconnections=7
currentconnections=1
maxreachedconnections=1
opsinitiated=16
opscompleted=15
abandonsrequested=0
abandonscompleted=0
addsrequested=0
addscompleted=0
bindsrequested=7
bindscompleted=7
```

comparesrequested=0
comparescompleted=0
deletesrequested=0
deletescompleted=0
extopsrequested=0
extopscompleted=0
modifiesrequested=0
modifiescompleted=0
modifydnsrequested=0
modifydnscompleted=0
searchesrequested=5
searchescompleted=4
unbindsrequested=4
unbindscompleted=4
unknownopsrequested=0
unknownopscompleted=0
entriessent=19
bytessent=16523
searchreferencessent=0
currenttime=Mon Jul 12 18:22:56.255129 2010
starttime=Mon Jul 12 18:18:13.872139 2010
resetttime=Mon Jul 12 18:18:13.872139 2010
resets=0

cn=back_endcdbm,cn=monitor
namingcontexts=CN=CONFIGURATION
namingcontexts=CN=IBMPOLICIES
opsinitiated=0
opscompleted=0
abandonsrequested=0
abandonscompleted=0
addsrequested=0
addscompleted=0
bindsrequested=0
bindscompleted=0
comparesrequested=0
comparescompleted=0
deletesrequested=0
deletescompleted=0
extopsrequested=0
extopscompleted=0
modifiesrequested=0
modifiescompleted=0
modifydnsrequested=0
modifydnscompleted=0
searchesrequested=0
searchescompleted=0
unbindsrequested=0
unbindscompleted=0
unknownopsrequested=0
unknownopscompleted=0
entriessent=0
bytessent=0
searchreferencessent=0
filter_cache_size=5000

filter_cache_current=9
filter_cache_hit=0
filter_cache_miss=9
filter_cache_percent_hit=0.00%
filter_cache_refresh=0
filter_cache_refresh_avgsz=0
filter_cache_bypass_limit=100

cn=back endGDBML,cn=monitor
namingcontexts=CN=CHANGELOG
opsinitiated=0
opscompleted=0
abandonsrequested=0
abandonscompleted=0
addsrequested=0
addscompleted=0
bindsrequested=0
bindscompleted=0
comparesrequested=0
comparescompleted=0
deletesrequested=0
deletescompleted=0
extopsrequested=0
extopscompleted=0
modifiesrequested=0
modifiescompleted=0
modifydnsrequested=0
modifydnscompleted=0
searchesrequested=0
searchescompleted=0
unbindsrequested=0
unbindscompleted=0
unknownopsrequested=0
unknownopscompleted=0
entriessent=0
bytessent=0
searchreferencessent=0
filter_cache_size=0
filter_cache_current=0
filter_cache_hit=0
filter_cache_miss=0
filter_cache_percent_hit=0.00%
filter_cache_refresh=0
filter_cache_refresh_avgsz=0
filter_cache_bypass_limit=100

cn=back endLDBMDB1,cn=monitor
namingcontexts=0=IBM,C=US
opsinitiated=0
opscompleted=0
abandonsrequested=0
abandonscompleted=0
addsrequested=0
addscompleted=0
bindsrequested=0


```
bindscompleted=0
comparesrequested=0
comparescompleted=0
deletesrequested=0
deletescompleted=0
extopsrequested=0
extopscompleted=0
modifiesrequested=0
modifiescompleted=0
modifydnsrequested=0
modifydnscompleted=0
searchesrequested=0
searchescompleted=0
unbindsrequested=0
unbindscompleted=0
unknownopsrequested=0
unknownopscompleted=0
entriessent=0
bytessent=0
searchreferencessent=0
filter_cache_size=5000
filter_cache_current=0
filter_cache_hit=0
filter_cache_miss=0
filter_cache_percent_hit=0.00%
filter_cache_refresh=0
filter_cache_refresh_avgsz=0
filter_cache_bypass_limit=100

cn=back endSDBM-0001,cn=monitor
namingcontexts=CN=RACF,0=IBM,C=IN
opsinitiated=0
opscompleted=0
abandonsrequested=0
abandonscompleted=0
addsrequested=0
addscompleted=0
bindsrequested=0
bindscompleted=0
comparesrequested=0
comparescompleted=0
deletesrequested=0
deletescompleted=0
extopsrequested=0
extopscompleted=0
modifiesrequested=0
modifiescompleted=0
modifydnsrequested=0
modifydnscompleted=0
searchesrequested=0
searchescompleted=0
unbindsrequested=0
unbindscompleted=0
unknownopsrequested=0
unknownopscompleted=0
```

entriessent=0
bytessent=0
searchreferencessent=0

cn=back endMonitor,cn=monitor
namingcontexts=CN=MONITOR
opsinitiated=5
opscompleted=4
abandonsrequested=0
abandonscompleted=0
addsrequested=0
addscompleted=0
bindsrequested=0
bindscompleted=0
comparesrequested=0
comparescompleted=0
deletesrequested=0
deletescompleted=0
extopsrequested=0
extopscompleted=0
modifiesrequested=0
modifiescompleted=0
modifydnsrequested=0
modifydnscompleted=0
searchesrequested=5
searchescompleted=4
unbindsrequested=0
unbindscompleted=0
unknownopsrequested=0
unknownopscompleted=0
entriessent=24
bytessent=21361
searchreferencessent=0

cn=back endSchema,cn=monitor
namingcontexts=CN=SCHEMA
opsinitiated=0
opscompleted=0
abandonsrequested=0
abandonscompleted=0
addsrequested=0
addscompleted=0
bindsrequested=0
bindscompleted=0
comparesrequested=0
comparescompleted=0
deletesrequested=0
deletescompleted=0
extopsrequested=0
extopscompleted=0
modifiesrequested=0
modifiescompleted=0
modifydnsrequested=0
modifydnscompleted=0
searchesrequested=0

searchescompleted=0
unbindsrequested=0
unbindscompleted=0
unknownopsrequested=0
unknownopscompleted=0
entriessent=0
bytessent=0
searchreferencessent=0
dn_cache_size=1000
dn_cache_current=70
dn_cache_hit=40
dn_cache_miss=69
dn_cache_percent_hit=36.70%
dn_cache_refresh=0
dn_cache_refresh_avgsz=0

cn=back_endRootDSE,cn=monitor

opsinitiated=0
opscompleted=0
abandonsrequested=0
abandonscompleted=0
addsrequested=0
addscompleted=0
bindsrequested=0
bindscompleted=0
comparesrequested=0
comparescompleted=0
deletesrequested=0
deletescompleted=0
extopsrequested=0
extopscompleted=0
modifiesrequested=0
modifiescompleted=0
modifydnsrequested=0
modifydnscompleted=0
searchesrequested=0
searchescompleted=0
unbindsrequested=0
unbindscompleted=0
unknownopsrequested=0
unknownopscompleted=0
entriessent=0
bytessent=0
searchreferencessent=0

cn=operations,cn=monitor

searchStats=ldap:///CN=_v??sub?(objectclass=*)?success,numOps=1,avg=278,rate=0,maxRate=0,maxRateTimeStamp=20100712182226.346490Z,createTimeStamp=20100712182226.346490Z,ID=2
searchStats=ldap:///CN=_v??base?(objectclass=*)?success,numOps=2,avg=94,rate=0,maxRate=1,maxRateTimeStamp=20100712182132.327366Z,createTimeStamp=20100712181933.177832Z,ID=0
searchStats=ldap:///CN=_v??one?(objectclass=*)?success,numOps=1,avg=243,rate=0,maxRate=1,maxRateTimeStamp=20100712182002.069496Z,createTimeStamp=20100712182002.069496Z,ID=1

```
currenttimestamp=20100712182256.255319Z
resettimestamp=20100712181814.466217Z
resets=0
numtrimmed=0
entries=3
cachesize=1000
```

13.2 Monitoring and managing advanced replication

This topic discusses the use of the z/OS IBM Tivoli Directory Server client utilities to check the current advanced replication status for each of your configured replication agreements.

13.2.1 Showing advanced replication configuration information:

All replication related information is available using `ldapsearch` command. To retrieve replication topology information from all configured suffixes, you can do a subtree scope search with NULL base, and filter as `objectclass=ibm-repl*`. All replication related data is stored with objectclass of type `ibm-repl*` such as `ibm-replicationContext`, `ibm-replicasubentry`, and many more.

```
ldapsearch -L -h ldaphost -p port -D adminDN -w passwd -s sub -b ""
objectclass=ibm-repl*
```

The `-L` option displays data in LDIF format.

Example:

```
ldapsearch -L -D cn=root -w root -p 13389 -s sub -b "" objectclass=ibm-repl*
```

You should receive output similar to the following:

```
dn: o=ibm, c=us
objectclass: top
objectclass: organization
objectclass: ibm-replicationContext
o: ibm
```

```
dn: ibm-replicaGroup=default, o=ibm, c=us
objectclass: top
objectclass: ibm-replicaGroup
ibm-replicagroup: default
```

```
dn: ibm-replicaServerId=ldap8103,ibm-replicaGroup=default, o=ibm, c=us
objectclass: top
objectclass: ibm-replicaSubentry
ibm-replicaserverid: ldap8103
ibm-replicationserverismaster: true
cn: Peer2
description: Subentry for Peer2.
```

```
dn: ibm-replicaServerId=ldap8003,ibm-replicaGroup=default, o=ibm, c=us
objectclass: top
objectclass: ibm-replicaSubentry
ibm-replicaserverid: ldap8003
ibm-replicationserverismaster: true
```

```

cn: Peer1
description: Subentry for Peer1.

dn: cn=Peer1, ibm-replicaServerId=ldap8103,ibm-replicaGroup=default,o=ibm,c=us
objectclass: top
objectclass: ibm-replicationAgreement
cn: Peer1
ibm-replicaconsumerid: ldap8003
ibm-replicaurl: ldap://WTSC80.ITS0.IBM.COM:13389
ibm-replicacredentialsdn: cn=ReplicaBindCredentials, o=ibm, c=us
description: Replication agreement from peer2 to peer1.

dn: cn=Peer2, ibm-replicaServerId=ldap8003,ibm-replicaGroup=default,o=ibm,c=us
objectclass: top
objectclass: ibm-replicationAgreement
cn: Peer2
ibm-replicaconsumerid: ldap8103
ibm-replicaurl: ldap://WTSC81.ITS0.IBM.COM:13389
ibm-replicacredentialsdn: cn=ReplicaBindCredentials, o=ibm, c=us
description: Replication agreement from peer1 to peer2.

dn: cn=ReplicaBindCredentials, o=ibm, c=us
objectclass: ibm-replicationCredentialsSimple
objectclass: ibm-replicationcredentials
objectclass: top
cn: ReplicaBindCredentials
description: Bind Credentials on peer1 and peer2 to bind to each other.

```

Note: Output of the `ldapsearch` command can vary from one environment to another.

If the `ldapsearch` command does not return any result with correct input values, then advanced replication is not configured in the LDAP server.

Similarly, you can retrieve replication topology information from a specific configured suffix or specific replication context. You just need to change base to the configured suffix or replication context entry. Use the following `ldapsearch` command to do so:

```
ldapsearch -L -h ldaphost -p port -D adminDN -w passwd -s sub -b "configured
suffix or replication context entry" objectclass=ibm-repl*
```

The `-L` option displays data in LDIF format.

For example:

```
ldapsearch -L -D cn=root -w root -p 13389 -s sub -b "o=ibm,c=us"
objectclass=ibm-repl*
```

You should receive output similar to the following:

```

dn: o=ibm, c=us
objectclass: top
objectclass: organization
objectclass: ibm-replicationContext
o: ibm

dn: ibm-replicaGroup=default, o=ibm, c=us

```

```
objectclass: top
objectclass: ibm-replicaGroup
ibm-replicagroup: default
```

```
dn: ibm-replicaServerId=ldap8103,ibm-replicaGroup=default, o=ibm, c=us
objectclass: top
objectclass: ibm-replicaSubentry
ibm-replicaserverid: ldap8103
ibm-replicationserverismaster: true
cn: Peer2
description: Subentry for Peer2.
```

```
dn: ibm-replicaServerId=ldap8003,ibm-replicaGroup=default, o=ibm, c=us
objectclass: top
objectclass: ibm-replicaSubentry
ibm-replicaserverid: ldap8003
ibm-replicationserverismaster: true
cn: Peer1
description: Subentry for Peer1.
```

```
dn: cn=Peer1, ibm-replicaServerId=ldap8103,ibm-replicaGroup=default,o=ibm,c=us
objectclass: top
objectclass: ibm-replicationAgreement
cn: Peer1
ibm-replicaconsumerid: ldap8003
ibm-replicaurl: ldap://WTSC80.ITS0.IBM.COM:13389
ibm-replicacredentialsdn: cn=ReplicaBindCredentials, o=ibm, c=us
description: Replication agreement from peer2 to peer1.
```

```
dn: cn=Peer2, ibm-replicaServerId=ldap8003,ibm-replicaGroup=default,o=ibm,c=us
objectclass: top
objectclass: ibm-replicationAgreement
cn: Peer2
ibm-replicaconsumerid: ldap8103
ibm-replicaurl: ldap://WTSC81.ITS0.IBM.COM:13389
ibm-replicacredentialsdn: cn=ReplicaBindCredentials, o=ibm, c=us
description: Replication agreement from peer1 to peer2.
```

```
dn: cn=ReplicaBindCredentials, o=ibm, c=us
objectclass: ibm-replicationCredentialsSimple
objectclass: ibm-replicationcredentials
objectclass: top
cn: ReplicaBindCredentials
description: Bind Credentials on peer1 and peer2 to bind to each other.
```

Note: Output of the `ldapsearch` command can vary from one environment to another.

If the `ldapsearch` command does not return any result with correct input values, then advanced replication is not configured in the given suffix entry.

The objects returned will include following:

- ▶ **objectclass: ibm-replicationContext:** The entry for the subtree that is replicated to other replica servers.

- ▶ **objectclass: ibm-replicaGroup:** The container for replication related configuration data.
- ▶ **objectclass: ibm-replicaSubentry:** These types of entries declare the servers that will be taking part in the replication topology. Each server participating in the topology can have one replica subentry, and replica subentries contain a server ID attribute and an indication of the role the server plays (**ibm-replicationServerIsMaster**).
- ▶ **objectclass: ibm-replicationAgreement:** These types of entries occur under replica subentries. When these entries appear under a specific server's replica subentry, they define a replication agreement from that server to another server in the topology.

Each replication agreement contains the following information:

- **ibm-replicaConsumerId:** The server ID of the consumer server.
- **ibm-replicaURL:** The LDAP URL of the consumer server.
- **ibm-replicaCredentialsDN:** The DN of the entry containing the credentials used to bind to the consumer.

Agreements can also contain the following:

- **ibm-replicaScheduleDN:** The DN of a schedule entry that determines when replication updates are sent to this consumer. If no schedule is specified, replication defaults to immediate mode.
- **ibm-replicationOnHold:** A boolean indicating that replication to this consumer is suspended (or not).
- **ibm-replicationExcludedCapability:** The values of this attribute list OIDs of features that the consumer does not support. Operations related to these capabilities are then excluded from the updates sent to this consumer.
- **ibm-replicationMethod:** Single threaded or multi-threaded.
- **ibm-replicationConsumerConnections:** For a replication agreement using the single-threaded replication method, the number of consumer connections is always one, so the attribute value is ignored. For an agreement using multi-threaded replication, the number of connections can be configured from 1 to 32. If no value is specified on the agreement, the number of consumer connections is set to one.
- **ibm-replicationWaitOnDependency:** Indicates whether the server will await the completion of the replication of dependencies prior to sending a replication update to a consumer.
- **ibm-replicationFilterDN:** A DN identifying the filter entry.

13.2.2 Extended operations related to advanced replication

A set of extended operations has been added to allow administrators to manage advanced replication (**1dapexop** was shipped in z/OS V1R11 for advanced replication). Using extended operations an administrator can achieve the following:

- ▶ Distribute configured replication topology to all consumers.
- ▶ Manage the replication queues.
- ▶ Manage any replication related errors.
- ▶ Manage the quiesce state of the replication context.
- ▶ Resume and suspend replication processing.

The **1dapexop** utility provides advanced replication extended operations. The **1dapexop** command line utility require the user to bind with the LDAP server administrator's DN and password.

The following advanced replication extended operations are available on the **ldapexop** utility:

- ▶ **repl topology**: This extended operation distributes advanced replication topology-related entries to all consumers. It then cascades this extended operation to all the consumers. This results in a cascading of the topology-related entries to all servers that participate in replication for a given replication context.
- ▶ **controlqueue**: Skips one or all pending changes in the advanced replication queue.
- ▶ **controlrepl**: Suspends or resumes all advanced replication-related activity.
- ▶ **cascrepl**: Allows you to quiesce, unquiesce, or force immediate replication of all pending changes. When the extended operation is performed on the supplier that this extended operation was issued against, it proceeds to cascade the extended operation to one or all of its consumers.
- ▶ **controlreplerr**: Deletes, retries, or shows any of the failed replication operations that resulted by an unsuccessful return code returned to the supplier from the consumer.
- ▶ **quiesce**: Allows you to quiesce or unquiesce a replication context.

13.2.3 Monitoring advanced replication status

In z/OS IBM Tivoli Directory Server, an LDAP administrator can monitor the advanced replication processing and troubleshoot problems using LDAP search requests to retrieve operational attributes available for the roots of the replication contexts and replication agreements.

Operational attributes of **ibm-replicationContext**:

Operational attributes for the replication context can be retrieved using the following **ldapsearch** command:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""  
objectclass=ibm-replicationContext attribute
```

Here *attribute* needs to be replaced with any of the following operational attributes:

- ▶ **ibm-replicationThisSeverIsMaster**: Using this boolean operational attribute one can find whether the server is the master of the replication context. If set to true, the server is the master of the replication context. If set to false, the server is not the master of the replication context. Use the following **ldapsearch** command to retry **ibm-replicationThisSeverIsMaster** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""  
objectclass=ibm-replicationContext ibm-replicationThisSeverIsMaster
```

- ▶ **ibm-replicationIsQuiesced**: Using this boolean operational attribute one can find whether the replication context is quiesced. If set to true, the replication context is quiesced. If set to false, the replication context is not quiesced. Use the following **ldapsearch** command to retry **ibm-replicationIsQuiesced** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""  
objectclass=ibm-replicationContext ibm-replicationIsQuiesced
```


Operational attributes of ibm-replicationAgreement:

Operational attributes for the replication agreement can be retrieved using the following **ldapsearch** command:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""  
objectclass=ibm-replicationAgreement attribute
```

Here *attribute* needs to be replaced with any of the following operational attributes:

- ▶ **ibm-replicationChangeLDIF**: Retrieves the LDIF representation of the next pending change that has not yet been replicated and has resulted in advanced replication being stalled to the consumer server. If there is not a stalled replication change, the value is N/A. Use the following **ldapsearch** command to retry the **ibm-replicationChangeLDIF** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""  
objectclass=ibm-replicationAgreement ibm-replicationChangeLDIF
```

- ▶ **ibm-replicationFailedChangeCount**: Displays the number of advanced replication operations that have failed in this replication agreement. This number is shared among all replication agreement entries on the back end level by the **ibm-slapedRep1MaxErrors** attribute in the CDBM back end configuration entry `cn=Replication, cn=Configuration`. Use the following **ldapsearch** command to retry the **ibm-replicationFailedChangeCount** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""  
objectclass=ibm-replicationAgreement ibm-replicationFailedChangeCount
```

- ▶ **ibm-replicationFailedChanges**: Lists all the logged replication operations that have failed. The number of attribute values is shared among all replication agreement entries on the back end level by the **ibm-slapedRep1MaxErrors** attribute in the CDBM back end configuration entry `cn=Replication, cn=Configuration`. Use the following **ldapsearch** command to retry the **ibm-replicationFailedChanges** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""  
objectclass=ibm-replicationAgreement ibm-replicationFailedChanges
```

- ▶ **ibm-replicationLastActivationTime**: Retrieves the Zulu format time stamp when advanced replication actively began replicating queued updates. Use the following **ldapsearch** command to retry the **ibm-replicationLastActivationTime** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""  
objectclass=ibm-replicationAgreement ibm-replicationLastActivationTime
```

- ▶ **ibm-replicationLastChangeID**: Retrieves the replication change ID of the last successfully completed advanced replication update. Use the following **ldapsearch** command to retry the **ibm-replicationLastChangeID** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""  
objectclass=ibm-replicationAgreement ibm-replicationLastChangeID
```

- ▶ **ibm-replicationLastFinishTime**: Retrieves the Zulu format time stamp when advanced replication updates in the queue were completed and the server began to wait for more operations to appear in the advanced replication queue. Use the following **ldapsearch** command to retry the **ibm-replicationLastFinishTime** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""  
objectclass=ibm-replicationAgreement ibm-replicationLastFinishTime
```

- ▶ **ibm-replicationLastResult**: Displays the result from the last advanced replication operation or connection attempt to a consumer server. Use the following **ldapsearch** command to retry the **ibm-replicationLastResult** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""  
objectclass=ibm-replicationAgreement ibm-replicationLastResult
```

- ▶ **ibm-replicationLastResultAdditional**: Displays the descriptive message that supplements the return code message from the last replication attempt. Use the following **ldapsearch** command to retry the **ibm-replicationLastResultAdditional** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""
objectclass=ibm-replicationAgreement ibm-replicationLastResultAdditional
```

- ▶ **ibm-replicationNextTime**: Displays the Zulu format time stamp of the next time advanced replication would begin if pending changes existed. Use the following **ldapsearch** command to retry the **ibm-replicationNextTime** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""
objectclass=ibm-replicationAgreement ibm-replicationNextTime
```

- ▶ **ibm-replicationPendingChangeCount**: Displays the number of replication operations that are waiting to be replicated to a consumer server. Use the following **ldapsearch** command to retry the **ibm-replicationPendingChangeCount** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""
objectclass=ibm-replicationAgreement ibm-replicationPendingChangeCount
```

- ▶ **ibm-replicationPendingChanges**: Lists all changes waiting to be replication to a consumer server. Use the following **ldapsearch** command to retry the **ibm-replicationPendingChanges** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""
objectclass=ibm-replicationAgreement ibm-replicationPendingChanges
```

- ▶ **ibm-replicationState**: Displays the current state of the advanced replication queue. It has one of the following values:

- **active**: Indicates that advanced replication is occurring from this replication agreement.
- **binding**: Indicates that the replication agreement is in the process of authenticating with the consumer server.
- **connecting**: Indicates that the replication agreement is attempting to contact the consumer server.
- **on hold**: Indicates that the replication agreement is on hold.
- **ready**: Indicates immediate replication mode, ready to send updates as they occur.
- **retrying**: Indicates that the server is retrying the current change every 60 seconds until it succeeds.
- **suspended**: Indicates that the replication agreement is suspended.
- **waiting**: Indicates that the replication agreement is currently waiting for the next scheduled replication to occur.

Use the following **ldapsearch** command to retry the **ibm-replicationState** attribute:

```
ldapsearch -h ldaphost -p port -D adminDN -w passwd -s sub -b ""
objectclass=ibm-replicationAgreement ibm-replicationState
```

13.3 Using activity logging

The activity log keeps a record of the add, compare, delete, extended operation, modify, modifydn, and search requests that were attempted and performed in the LDAP server. The activity log also shows the time stamp of when a client binds and unbinds from the directory, and all other operations it is configured to log. These measurements allow the administrator to identify LDAP operations that take a long time to complete.

The system administrator can use the activities stored in the activity log to check for suspicious patterns of activity that might indicate security violations. If security is violated, the activity log can be used to determine how and when the problem occurred, and perhaps the amount of damage done. This information is useful both for recovery from the violation and for development of better security measures to prevent future problems.

The LDAP server activity logging support has a number of features that allow the LDAP administrator to customize the client activity to be logged. These features include:

- ▶ Logging the start or end of a client operation.
- ▶ Logging only client update operations (add, delete, modify, extended operations, and modifydn).
- ▶ Logging all client operations (add, bind, compare, delete, extended operations, modify, modifydn, search, and unbind).
- ▶ Logging messages generated by the server.
- ▶ Logging hourly client activity summary statistics.
- ▶ Logging only requests from certain client IP addresses.
- ▶ Activity log file archiving or rollover that copies the current activity log file to another location for load analysis.

Activity log configuration

The **logfile** configuration option in the global section of the LDAP server configuration file specifies the location of the z/OS UNIX System Services file or dataset where activity log records are written. If a z/OS UNIX System Services file is specified for the logfile configuration option, a fully qualified name must be specified.

For example:

```
logfile /u/npatel/logs/ldap.activity.log
```

Note: If the location is the z/OS UNIX System Services file, the LDAP instance owner must have read and write permission to the specified location and file.

The LDAP server supports automatic activity log file rollover or archiving based on the time of day or the size of the log file. The activity log archiving is only supported when the log file is a z/OS UNIX System Services file or a GDG (Generated Data Group) dataset.

- ▶ When the **logFileRolloverTOD** configuration option has a value between 00:00 and 23:59, it indicates the time each day when the current activity log file is archived.
- ▶ When the **logFileRolloverSize** configuration option has a non-zero size, it indicates the size in bytes, megabytes, kilobytes, or gigabytes that the activity log file is required to have reached before it is archived or rolled over.

If the logfile is a z/OS UNIX System Services file, and the activity log file reaches one of these thresholds, the following occur:

- ▶ The current activity log file is renamed with the current Zulu time stamp appended to the end of the file name.
- ▶ If the **logFileRolloverDirectory** configuration option is specified, then the archived log file is moved to that directory. Otherwise the archived activity log file is left in the same directory as the current activity log file.

If the **logfile** and **logFileRolloverDirectory** configuration options both specify a Generated Data Group (GDG) dataset, the current activity log file is closed and a new dataset generation

is created in the base specified by the **logFileRolloverDirectory** configuration option. The new dataset generation is used for the activity log file until the next rollover or archiving occurs.

Note: Activity log file rollover or archiving is not supported when the **logfile** or **logFileRolloverDirectory** configuration options specify a partitioned dataset, sequential dataset, or a DD card.

By default the z/OS LDAP server records all client operations if activity logging is enabled. However, there is a way to track specific clients from specific IP addresses by using the **logFileFilter** parameter in the configuration file. The default setting for the **logFileFilter** parameter allows all client activity to be logged in the activity log file. For example, the server can be configured to only log client requests from IP address 1.2.3.4 by specifying **logFileFilter (ibm-filterIP=1.2.3.4)**.

Note: Multiple LDAP servers writing to the same activity log is not supported. If running in multi-server mode and the same configuration file is shared among the LDAP servers, separate **logfile** and **logfileRolloverDirectory** configuration options can be specified in a system specific include file for each LDAP server.

Refer to chapter 9 of *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for more information about each of these steps.

13.4 Operations monitor

Using operations monitor, you can monitor statistics of search requests with predefined search patterns.

The global section's **operationsMonitor** configuration option determines which types of search patterns are monitored. The operations monitor supports two types of search patterns:

- **searchStats**

A **searchStats** pattern consists of the search parameters like search base, scope, filter, and attributes to be returned and status (success or failure). Output of this search pattern is useful in evaluating performance of search patterns. If **operationsMonitor** is set to **ipAny** (the default), then only **searchStats** patterns are monitored.

- **searchIPStats**

A **searchIPStats** pattern consists of client IP address along with the same elements as the **searchStats** pattern. Output of this search pattern is useful in tracking clients activity, which can be used in internal auditing. If **operationsMonitor** set to **ip**, then only **searchIPStats** patterns are monitored.

If **operationsMonitor** is set to **any**, the operations monitor monitors both **searchStats** and **searchIPStats** patterns.

13.5 Audit logging

Audit logging is used to improve the security of the directory server. If security is violated, the audit log can be used to determine how and when the problem occurred and perhaps the amount of damage done. IBM Tivoli Directory Server for z/OS server can be configured to generate SMF type-83 subtype three audit records. The SMF type-83 log records containing LDAP events can be unloaded by using the RACF SMF data **Unload** utility for further analysis using auditing tools. These audit records contain information provided on LDAP client operation requests. The IBM Tivoli Directory Server for z/OS server can be configured to write audit records when:

1. The operation successfully completes
2. The operation fails
3. For when an operation successfully completes and when an operation fails

Note: SMF type-83 subtype 3 audit records are not created by the LDAP server when a request is handled by a plug-in.

Auditing can be turned ON or OFF by editing the **audit** parameter. This parameter specifies which operations are to be audited and the associated audit level. This option can be specified multiple times, once to turn auditing on or off and one or more times for each audit level to specify the operations to audit for that level. Multiple operations can be specified for a level by either putting a + between them on the audit option, or by specifying multiple audit options with the same level.

Operations can be audited all the time or only when they fail. The following audit levels are supported:

none	An LDAP audit record is not generated for the specified operations
all	An LDAP audit record is generated for the specified operations.
error	An LDAP audit record is generated for the specified operations when they fail.

When the LDAP server is running, auditing can be turned on or off and the specifications of which operations are to be audited and their associated audit level can be changed using the LDAP server AUDIT operator modify command. The format of the AUDIT operator modify command is:

```
f ldap8001,audit on | off | all,operations | error,operations | none,operations
```

The supported values for operations can be one or more of **add**, **bind**, **compare**, **connect**, **delete**, **disconnect**, **exop**, **modify**, **modifydn**, **search**, and **unbind**. If an operation is specified in more than one level, the last level is used for the operation. If an operation is not specified in any level, the level defaults to none for that operation.

The current audit settings can be displayed using the following LDAP server DISPLAY operator modify command:

```
f ldap8001,display audit
```

Refer to appendix E of *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05 for more information about SMF records.



Debugging

This chapter discusses the various debugging and tracing methods provided by IBM Tivoli Directory Server for z/OS.

14.1 Overview

The process of configuring IBM Tivoli Directory Server using **dsconfig** command, for various reasons, is not always error free. The directory administrator can encounter problems with configuring various back ends, or the server might fail to start for no obvious reason.

Debugging is the process of finding the cause of the problem using various tools and techniques. z/OS IBM Tivoli Directory Server provides administrators with command line options, tools, and detailed log files that help find the cause of the problem.

14.2 Debugging problems

The following sections describe how to debug configuration problems, directory server errors, and directory server debug modes.

14.2.1 Debugging configuration problems

If the configuration fails, you have no choice but to resolve the issue. The basic steps towards making the IBM Tivoli Directory Server up and running are:

- ▶ Edit `ds.profile`, `ds.slapped.profile`, `ds.db2.profile` and `ds.racf.profile` files.
- ▶ Execute **dsconfig** command.

Editing profile files is fairly straightforward and is less error prone. However, any mistake in the editing leads error in `dsconfig` execution. If **dsconfig** fails, the following sources can be checked to find the cause of the failure:

1. Output

The **dsconfig** command is started from a console command line prompt. As the configuration progresses, status messages (and limited error messages) are displayed in the associated console window. If a problem occurs, the user should copy these messages to the system clipboard and then save them in a file for the support teams.

2. Trace file

If the above source is not sufficient for determining the cause of the problem, you can use **dsconfig** debugging. In **dsconfig** debugging, add option **-d** with the different debug levels. The best option is to use FULL tracing level (**-d 2147483647** or **-d ALL**). The **dsconfig** trace output can be routed to a file to save for later perusal, as shown in the following example.

```
dsconfig -i ds.profile -d ALL -a yes > /tmp/dsconfig.output 2>&1
```

14.2.2 Using server debug modes

If the logs do not provide enough information to resolve a problem, you can run IBM Tivoli Directory Server in a special debug mode that generates detailed information. The LDAP server writes messages to **stdout** and **stderr**. Messages sent to **stdout** and **stderr** appear in DD:DSOUT in the provided JCL when running the LDAP server. Output from the LDAP server debug facility is directed to the file specified by the `LDAP_DEBUG_FILENAME` environment variable. If this environment variable is not set, the output is sent to **stdout**, which is redirected to DSOUT as explained above.

A table containing the various debug levels is provided in *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05.

In z/OS IBM Tivoli Directory Server, the debug level for the server can be set at a number of levels:

1. Using LDAP_DEBUG environment variable:

Before starting LDAP server you can enable trace tracing by exporting the LDAP_DEBUG variable to different available trace levels.

Example:

```
export LDAP_DEBUG=2147483647
export LDAP_DEBUG_FILE=/tmp/trace.file
```

2. The -d parameter while starting the server

The debug level specified on this parameter replaces, adds to, or deletes from the preceding debug level:

a. Replaces

The following example replaces the current debug level that is off or has been set by the LDAP_DEBUG environment variable with the new debug level of only ERROR.

```
s ldap8001,parms='-d ERROR'
```

b. Adds

The following example adds the ERROR debug level to the current debug level that is off or has been set by the LDAP_DEBUG environment variable.

```
s ldap8001,parms='-d +ERROR'
```

c. Deletes

The following example removes the ERROR debug level from the current debug level that is off or has been set by the LDAP_DEBUG environment variable.

```
s ldap8001,parms='-d -ERROR'
```

3. Dynamic debugging

Use this facility when the LDAP server is running to dynamically turn the debugging facility on and off. You can also replace the current debug levels, add to the current debug levels, or remove current debug levels. The following command can be sent to the LDAP server from the SDSF or the operator's console.

Note: If the command is entered from SDSF, it must be preceded by a slash (/)

```
f ldap8001,debug debug_level
```

Example:

```
f ldap8001,debug ALL
```

The *debug_level* is a mask that specifies the needed debug level. Debug information is added to the output associated with the LDAP server.

14.2.3 Using CTRACE in-memory records

CTRACE is a component trace that provides an in-memory tracing interface common with other z/OS products.

The LDAP CTRACE support captures the following LDAP server output:

- ▶ All messages
- ▶ All debug message output for the ERROR debug level, regardless of debug level setting

- ▶ All debug message output for active debug levels.

By always capturing ERROR debug output, the CTRACE provides FFDC First Failure Data Capture to assist in problem debugging without the performance overhead required by running with the debug facility enabled.

When the debug facility is active, debug output by default goes to both the CTRACE in-memory table and the output file indicated by the LDAP_DEBUG_FILENAME environment variable (or **stdout** if the environment variable is not set). If LDAP_DEBUG_FILENAME=CTRACE_ONLY is set, the debug output is only sent to CTRACE. The LDAP server DEBUG operator modify command can be used when the LDAP server is running to change whether debug output is sent only to CTRACE, or to both CTRACE and the debug output file.

To direct debug output to both CTRACE and to the debug output file:

```
f dssrv,debug output=both
```

To direct debug output to CTRACE only:

```
f dssrv,debug output=memory
```

The LDAP CTRACE support is initialized during LDAP server startup. If CTRACE initialization fails, an error message is issued and server startup continues without CTRACE support. The LDAP CTRACE support cannot be turned off.

The default size of the storage used for the in-memory trace table is 1,024,000 bytes, which holds about 8000 CTRACE entries. The table wraps when the end is encountered, overwriting the oldest CTRACE entries. The trace table size can be increased by setting the LDAP_CTRACE_BUFFSIZE environment variable before the LDAP server is started.

Note: There is no maximum trace table size except that imposed by the availability of system storage.



Part 4

Appendixes



A

Sample plug-in code

This appendix provides the source code for the `plugin_sample.c` example.

Note: The code supplied here has not been subjected to any formal IBM test and is distributed on an “AS IS” basis without any warranty either express or implied. The implementation of any of the techniques described or used herein is a customer responsibility and depends on the customer’s operational environment. Although each item might have been reviewed for accuracy in a specific situation and run in a specific environment, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Source code for plugin_sample.c

The following is the plugin_sample.c code found in the /usr/lpp/ldap/examples directory.

```
??=ifdef __COMPILER_VER__
 2  ??=pragma filetag ("IBM-1047")
 3  ??=endif
 4
 5 /*****
 6 /* THIS FILE CONTAINS SAMPLE CODE.  IBM PROVIDES THIS CODE ON AN      */
 7 /* 'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS      */
 8 /* OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES  */
 9 /* OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.          */
10 /*****/
11
12 /*
13 *  plugin_sample.c: Post-Operation Sample Plug-in
14 *
15 *  This sample plug-in will test various SLAPI API routines.  The
16 *  following C code creates a post-operation plug-in that logs
17 *  LDAP server BIND requests to a specified file.
18 *
19 *  The compiled module needs to be located in a library in the load
20 *  list for the LDAP server, APF authorized and activated in the LDAP
21 *  server configuration file by adding the following option:
22 *
23 *      plugin postOperation PLUGSAMP plugin_init "auditFilename"
24 *
25 *      where auditFilename is the name of the file you wish to have
26 *      the log records written to; it must be within double
27 *      quotes.
28 *
29 *  Restart the LDAP server.  Use the debug parameter "PLUGIN" to
30 *  write sample plug-in trace messages to the LDAP server job log.
31 *
32 *      example: START LDAPSRV,PARMS='-d PLUGIN'
33 *      where LDAPSRV is an example name and represents the name
34 *      of your LDAP server start-up procedure.
35 *
36 *  Once started, browse your LDAP server job log for plug-in
37 *  initialization messages.  Also, the plug-in creates an
38 *  empty audit file.  Verify it was created.
39 *
40 *  To test, perform an LDAP operation, binding to the LDAP server.
41 *  The plug-in will log the event in your audit file.  The log record
42 *  will include the result code of the bind operation and the bind DN.
43 *
44 *      example: Result: 0 DN: GoodDN
45 *              or : Result: 49 DN: BadDN
46 *
47 */
48
```

Figure A-1 plugin_sample.c code

```

#pragma export(plugin_init)
50
51 #include <stdio.h>
52 #include <errno.h>
53 #include <iconv.h>
54 #include <slapi-plugin.h>
55
56 /*
57  * Local plug-in functions
58  */
59 static int plugin_bind_fn(Slapi_PBlock * pb);
60 static void plugin_close_fn(Slapi_PBlock * pb);
61
62 /*
63  * Plug-in private data area
64  */
65 typedef struct _plugin_private {
66     char                eyeCatcher[4];           /* "PLUG" */
67     pthread_mutex_t    pluginMutex;           /* Plugin mutex */
68     iconv_t            networkToLocal;        /* UTF-8 to IBM-1047 */
69     iconv_t            localToNetwork;       /* IBM-1047 to UTF-8 */
70     char *             auditFilename;        /* Audit filename (EBCDIC) */
71     FILE *             auditFile;           /* Audit file handle */
72 } plugin_private;
73
74
75 /*-----*/
76 /* This routine is called when the LDAP server initializes */
77 /* plug-in. */
78 /*-----*/
79 int plugin_init (
80     Slapi_PBlock *   pb)
81 {
82     int                rc=0;
83     int                argc;
84     char              **argv;
85     int                type;
86     plugin_private    *pdata;
87     char              *instr, *outstr;
88     size_t            inlth, outlth;
89
90     SLAPI_TRACE((LDAP_DEBUG_PLUGIN, "PLUGSAMP", "Entered.));
91

```

Figure A-2 *plugin_sample.c* code

```

92     /*
93     * Get the plug-in type
94     */
95     rc = slapi_pblock_get(pb, SLAPI_PLUGIN_TYPE, &type);
96     if (rc != 0) {
97         slapi_log_error(LDAP_MSG_HIGH, "PLUGSAMP",
98             "Unable to get plug-in type: Error %d\n",errno);
99         return -1;
100    }
101
102    if (type != SLAPI_PLUGIN_POSTOPERATION) {
103        slapi_log_error(LDAP_MSG_HIGH, "PLUGSAMP",
104            "This sample only accepts a post-operation type plug-in.\n");
105        return -1;
106    }
107
108    /*
109    * Get the count of plug-in initialization parameters
110    */
111    rc = slapi_pblock_get(pb, SLAPI_PLUGIN_ARGC, &argc);
112    if (rc != 0) {
113        slapi_log_error(LDAP_MSG_HIGH, "PLUGSAMP",
114            "Unable to get count of initialization parameters: Error %d\n",errno);
115        return -1;
116    }
117
118    if (argc != 1) {
119        slapi_log_error(LDAP_MSG_HIGH, "PLUGSAMP",
120            "Post-operation parameter count %d is incorrect.\n", argc);
121        return -1;
122    }
123
124    /*
125    * Get the plug-in initialization parameters
126    */
127    rc = slapi_pblock_get(pb, SLAPI_PLUGIN_ARGV, &argv);
128    if (rc != 0) {
129        slapi_log_error(LDAP_MSG_HIGH, "PLUGSAMP",
130            "Unable to get initialization parameters: Error %d\n",errno);
131        return -1;
132    }
133    SLAPI_TRACE((LDAP_DEBUG_PLUGIN, "PLUGSAMP", "Audit filename is \"%W\".", argv[0]));

```

Figure A-3 *plugin_sample.c* code


```

134
135     /*
136     * Register the server termination function
137     */
138     rc = slapi_pblock_set(pb, SLAPI_PLUGIN_CLOSE_FN, (void *)plugin_close_fn);
139     if (rc != 0) {
140         slapi_log_error(LDAP_MSG_HIGH, "PLUGSAMP",
141             "Unable to register CLOSE function: error %d.\n", errno);
142         return -1;
143     }
144
145     /*
146     * Register the BIND function
147     */
148     rc = slapi_pblock_set(pb, SLAPI_PLUGIN_BIND_FN, (void *)plugin_bind_fn);
149     if (rc != 0) {
150         slapi_log_error(LDAP_MSG_HIGH, "PLUGSAMP",
151             "Unable to register BIND function: error %d.\n", errno);
152         return -1;
153     }
154
155     /*
156     * Allocate the plug-in private data area, then initialize
157     */
158     pdata = (plugin_private *)slapi_ch_malloc(sizeof(plugin_private));
159     if (pdata == NULL) {
160         slapi_log_error(LDAP_MSG_HIGH, "PLUGSAMP",
161             "Unable to allocate plug-in private data area.\n");
162         return -1;
163     }
164
165     memset(pdata, 0, sizeof(plugin_private));
166     memcpy(pdata->eyeCatcher, "PLUG", 4);
167     pdata->networkToLocal = (iconv_t)(-1);
168     pdata->localToNetwork = (iconv_t)(-1);
169
170     /*
171     * Set up a mutex
172     */
173     rc = pthread_mutex_init(&pdata->pluginMutex, NULL);
174     if (rc != 0) {
175         slapi_log_error(LDAP_MSG_HIGH, "PLUGSAMP",
176             "Unable to create plugin mutex: Error %d\n", errno);
177         slapi_ch_free(pdata);
178         return -1;
179     }
180

```

Figure A-4 *plugin_sample.c* code

```

181     /*
182     * Set up UTF-8 to IBM-1047 converter
183     */
184     pdata->networkToLocal = iconv_open("IBM-1047", "UTF-8");
185     if (pdata->networkToLocal == (iconv_t)(-1)) {
186         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP",
187             "Unable to open UTF-8 to IBM-1047 converter: Error %d\n", errno);
188         goto cleanup;
189     }
190
191     /*
192     * Set up IBM-1047 to UTF-8 converter
193     */
194     pdata->localToNetwork = iconv_open("UTF-8", "IBM-1047");
195     if (pdata->localToNetwork == (iconv_t)(-1)) {
196         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP",
197             "Unable to open IBM-1047 to UTF-8 converter: Error %d\n", errno);
198         goto cleanup;
199     }
200
201     /*
202     * Open the audit file
203     */
204     inlth = strlen(argv[0]);
205     outlth = inlth;
206     pdata->auditFilename = slapi_ch_malloc(outlth+1);
207     if (pdata->auditFilename == NULL) {
208         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP",
209             "Insufficient storage available");
210         goto cleanup;
211     }
212
213     instr = argv[0];
214     outstr = pdata->auditFilename;
215     iconv(pdata->networkToLocal, &instr, &inlth, &outstr, &outlth);
216     *outstr = 0x00;
217
218     pdata->auditFile = fopen(pdata->auditFilename, "a");
219     if (pdata->auditFile == NULL) {
220         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP",
221             "Unable to open audit file '%s' for append: %s\n",
222             pdata->auditFilename, strerror(errno));
223         goto cleanup;
224     }
225

```

Figure A-5 *plugin_sample.c* code

```

226     /*
227     * Remember the private area
228     */
229     rc = slapi_pblock_set(pb, SLAPI_PLUGIN_PRIVATE, &pdata);
230     if (rc != 0) {
231         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP",
232             "Unable to set plugin private data address: Error %d\n", errno);
233         goto cleanup;
234     }
235
236     SLAPI_TRACE((LDAP_DEBUG_PLUGIN, "PLUGSAMP", "Exit rc=%d",rc));
237
238     return rc;
239
240
241     /*
242     * Clean up following an initialization failure
243     */
244 cleanup:
245     if (pdata->auditFile != NULL)
246         fclose(pdata->auditFile);
247
248     if (pdata->auditFilename != NULL)
249         slapi_ch_free(pdata->auditFilename);
250
251     if (pdata->localToNetwork != (iconv_t)(-1))
252         iconv_close(pdata->localToNetwork);
253
254     if (pdata->networkToLocal != (iconv_t)(-1))
255         iconv_close(pdata->networkToLocal);
256
257     pthread_mutex_destroy(&pdata->pluginMutex);
258     memset(pdata, 0, sizeof(plugin_private));
259     slapi_ch_free(pdata);
260     return -1;
261 }
262
263
264 /*-----*/
265 /* This routine is called when a BIND request has been processed. */
266 /*-----*/
267 static int plugin_bind_fn (
268     Slapi_PBlock * pb)
269 {
270     plugin_private *pdata=NULL;
271     int rc, resultCode, i;
272     char *bindDN, **groupList, **dnList, *cnvName=NULL;
273     char *instr, *outstr;
274     size_t inlth, outlth;

```

Figure A-6 *plugin_sample.c* code

```

275
276     SLAPI_TRACE((LDAP_DEBUG_PLUGIN, "PLUGSAMP", "Entered.));
277
278     /*
279     * Get the address of our private area
280     */
281     rc = slapi_pblock_get(pb, SLAPI_PLUGIN_PRIVATE, &pdata);
282     if (rc != 0) {
283         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP",
284             "Unable to get address of private area: Error %d\n", errno);
285         goto cleanup;
286     }
287
288     if (pdata == NULL || memcmp(pdata->eyeCatcher, "PLUG", 4) != 0) {
289         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP","Private area eyecatcher incorrect");
290         pdata = NULL;
291         goto cleanup;
292     }
293
294     /*
295     * Serialize access to our data structures
296     */
297     pthread_mutex_lock(&pdata->pluginMutex);
298
299     /*
300     * Get the BIND results
301     */
302     rc = slapi_pblock_get(pb, SLAPI_BIND_TARGET, &bindDN);
303     if (rc != 0) {
304         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP",
305             "Unable to get BIND target: Error %d\n", errno);
306         goto cleanup;
307     }
308
309     rc = slapi_pblock_get(pb, SLAPI_PLUGIN_OPRETURN, &resultCode);
310     if (rc != 0) {
311         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP",
312             "Unable to get result code: Error %d\n", errno);
313         goto cleanup;
314     }
315
316     rc = slapi_pblock_get(pb, SLAPI_REQUESTOR_GROUPS, &groupList);
317     if (rc != 0) {
318         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP",
319             "Unable to get group list: Error %d\n", errno);
320         goto cleanup;
321     }
322

```

Figure A-7 *plugin_sample.c* code

```

323     rc = slapi_pblock_get(pb, SLAPI_REQUESTOR_ALT_NAMES, &dnList);
324     if (rc != 0) {
325         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP",
326             "Unable to get alternate name list: Error %d\n", errno);
327         goto cleanup;
328     }
329
330     /*
331      * Log the BIND result
332      */
333     instr = bindDN;
334     inlth = strlen(bindDN);
335     outlth = inlth;
336     outstr = cnvName = slapi_ch_malloc(outlth+1);
337     if (cnvName == NULL) {
338         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP","Insufficient storage available");
339         goto cleanup;
340     }
341
342     outstr = cnvName;
343     iconv(pdata->networkToLocal, &instr, &inlth, &outstr, &outlth);
344     *outstr = 0x00;
345
346     rc = fprintf(pdata->auditFile, "Result: %d DN: %s\n", resultCode, cnvName);
347     if (rc < 0) {
348         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP","Unable to write to '%s': %s\n",
349             pdata->auditFilename, strerror(errno));
350         goto cleanup;
351     }
352
353     slapi_ch_free(cnvName);
354     cnvName = NULL;
355
356     if (groupList != NULL) {
357         rc = fprintf(pdata->auditFile, " Groups:\n");
358         if (rc < 0) {
359             slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP","Unable to write to '%s': %s\n",
360                 pdata->auditFilename, strerror(errno));
361             goto cleanup;
362         }
363     }

```

Figure A-8 *plugin_sample.c* code

```

364     for (i=0; groupList[i]!=NULL; i++) {
365         instr = groupList[i];
366         inlth = strlen(groupList[i]);
367         outlth = inlth;
368         outstr = cnvName = slapi_ch_malloc(outlth+1);
369         if (cnvName == NULL) {
370             slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP","Insufficient storage
available");
371             goto cleanup;
372         }
373
374         iconv(pdata->networkToLocal, &instr, &inlth, &outstr, &outlth);
375         *outstr = 0x00;
376
377         rc = fprintf(pdata->auditFile, "    %s\n", cnvName);
378         if (rc < 0) {
379             slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP","Unable to write to '%s': %s\n",
380                 pdata->auditFilename, strerror(errno));
381             goto cleanup;
382         }
383
384         slapi_ch_free(cnvName);
385         cnvName = NULL;
386     }
387 }
388
389 if (dnList != NULL) {
390     rc = fprintf(pdata->auditFile, " Alternate names:\n");
391     if (rc < 0) {
392         slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP","Unable to write to '%s': %s\n",
393             pdata->auditFilename, strerror(errno));
394         goto cleanup;
395     }
396
397     for (i=0; dnList[i]!=NULL; i++) {
398         instr = dnList[i];
399         inlth = strlen(dnList[i]);
400         outlth = inlth;
401         outstr = cnvName = slapi_ch_malloc(outlth+1);
402         if (cnvName == NULL) {
403             slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP","Insufficient storage
available");
404             goto cleanup;
405         }
406
407         iconv(pdata->networkToLocal, &instr, &inlth, &outstr, &outlth);
408         *outstr = 0x00;
409

```

Figure A-9 *plugin_sample.c* code

```

410         rc = fprintf(pdata->auditFile, "   %s\n", cnvName);
411         if (rc < 0) {
412             slapi_log_error(LDAP_MSG_HIGH,"PLUGSAMP","Unable to write to '%s': %s\n",
413                 pdata->auditFilename, strerror(errno));
414             goto cleanup;
415         }
416
417         slapi_ch_free(cnvName);
418         cnvName = NULL;
419     }
420 }
421
422     fflush(pdata->auditFile);
423
424     /*
425     * Clean up
426     */
427 cleanup:
428     if (pdata != NULL)
429         pthread_mutex_unlock(&pdata->pluginMutex);
430
431     if (cnvName != NULL)
432         slapi_ch_free(cnvName);
433
434     SLAPI_TRACE((LDAP_DEBUG_PLUGIN, "PLUGSAMP", "Exit."));
435
436     return 0;
437 }
438
439 /*-----*/
440 /* Plug-in termination function */
441 /*-----*/
442 static void plugin_close_fn (
443     Slapi_PBlock *    pb)
444 {
445     plugin_private    *pdata;
446
447     SLAPI_TRACE((LDAP_DEBUG_PLUGIN, "PLUGSAMP", "Entered."));
448
449     /*
450     * Release our private area
451     */
452     if (slapi_pblock_get(pb, SLAPI_PLUGIN_PRIVATE, &pdata) != 0) {
453         slapi_log_error(LDAP_MSG_HIGH,"TESTPLUG",
454             "Unable to get address of plugin private area: Error %d\n",  errno);

```

Figure A-10 *plugin_sample.c* code

```

455
456     } else if (pdata == NULL || memcmp(pdata->eyeCatcher, "PLUG", 4) != 0) {
457         slapi_log_error(LDAP_MSG_HIGH,"TESTPLUG", "Plugin private area eyecatcher
incorrect");
458
459     } else {
460
461         if (pdata->auditFile != NULL)
462             fclose(pdata->auditFile);
463
464         if (pdata->auditFilename != NULL)
465             slapi_ch_free(pdata->auditFilename);
466
467         if (pdata->localToNetwork != (iconv_t)(-1))
468             iconv_close(pdata->localToNetwork);
469
470         if (pdata->networkToLocal != (iconv_t)(-1))
471             iconv_close(pdata->networkToLocal);
472
473         pthread_mutex_destroy(&pdata->pluginMutex);
474         memset(pdata, 0, sizeof(plugin_private));
475         slapi_ch_free(pdata);
476     }
477
478     SLAPI_TRACE((LDAP_DEBUG_PLUGIN, "PLUGSAMP", "Exit."));
479 }

```

Figure A-11 *plugin_sample.c* code



Sample C code

This appendix provides a sample code in the C language used to recognize an expired password on a successful `ldap_bind` of a native authentication user and to change the password with an `ldap_modify`.

Note: The code supplied here has not been subjected to any formal IBM test and is distributed on an “AS IS” basis without any warranty either express or implied. The implementation of any of the techniques described or used herein is a customer responsibility and depends on the customer’s operational environment. Although each item might have been reviewed for accuracy in a specific situation and run in a specific environment, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Description of sample code

This Native Authentication and Expired Passwords API sample code is used to recognize an expired password on a successful `ldap_bind` of a native authentication user and to change the password with a `ldap_modify`.

The usage example assumes the data shown in Figure B-1.

```
dn:cn=someuser,c=ca
objectclass: person
objectclass: extensibleObject
cn:someuser
sn:lia
uid: SOMUSR
aclEntry: access-id:cn=someuser,c=ca:normal:rWSC:sensitive:rWSC:critical:rWSC

dn: racfid=SOMUSR,profiletype=user,cn=myracf
objectclass: racfuser
racfid: SOMUSR
objectclass: racfUser0mvsSegment
racf0mvsUid: 19
racfpassword: secret
```

Figure B-1 Existing data

For the existing native authentication user and its associated racf user, the sample code would produce the following if the racf user's password was expired (Figure B-2).

```
nlogin cn=someuser,c=ca secret

password expired, enter new password
superc
re-enter new password for verification
superc
password successfully changed
login successful for bindDN cn=someuser,c=ca
```

Figure B-2 Execution

The following is the source code in the C language:

```

#include <stdio.h>
#include <ldap.h>

static void usage( )
{
    fprintf( stderr, "Native Auth Login.\n" );
    fprintf( stderr, "usage:\n" );
    fprintf( stderr, "    nlogin bindDN currentpw\n");
    fprintf( stderr, "where:\n" );
    fprintf( stderr, "    bindDN is the distinguished name to login\n" );
    fprintf( stderr, "    currentpw is the current password\n" );
}

void main(int argc, char *argv[]) {

    char * host="9.12.47.80";
    int port=389;
    int rc;
    int parserc;
    LDAP *ld1;
    char * bindDn="cn=someuser,c=ca";
    char pw[9];
    char newPW[9];
    char verPW[9];
    int msg=LDAP_RES_ANY;
    LDAPMessage *result;
    LDAPControl **controls;
    static LDAPControl pwdPolicyCtl = {
        LDAP_PWDPOLICY_CONTROL_OID,      /* OID */
        { 0, NULL },                      /* no value */
        LDAP_OPT_OFF                       /* non-critical */
    };
    int ctrlerr;
    int ctrlwarn;
    int ctrlres;
    LDAPControl **servercontrol;
    int msgid = 0;

    char delVals[2][9];
    char addVals[2][9];
    LDAPMod delete_mod;
    LDAPMod add_mod;
    LDAPMod *pmods[3];

    if (argc<3) {
        usage();
        exit(1);
    }

    bindDn = argv[1];
    strcpy(pw,argv[2]);
}

```

Figure B-3 Code sample

```

/* create a connection to the ldap server */
if ( (ld1 = ldap_init(host,port)) == NULL) {
    fprintf(stderr,"ERROR\n");
    fprintf(stderr,"ldap_init failed. Check input parms\n");
    exit(5);
}

/* Set the password policy control to be sent on the bind */
/* indicating that we want password policy responses if present */
controls = (LDAPControl **)malloc( 2, sizeof(LDAPControl *));
controls[0] = &pwdPolicyCtl;
controls[1] = NULL;
rc = ldap_set_option(ld1,
                    LDAP_OPT_SERVER_CONTROLS,
                    controls);
if (rc != LDAP_SUCCESS) {
    fprintf( stderr, "ERROR\n");
    fprintf( stderr, "ldap_set_option failed rc=%d\n",rc);
    fprintf( stderr, "%s\n",ldap_err2string(ldap_get_errno(ld1)) );
    exit(5);
}

rc = ldap_simple_bind(ld1,bindDn,pw);
if (rc == -1)
{
    fprintf( stderr, "ERROR\n");
    fprintf( stderr, "ldap_simple_bind failed rc=%d\n",rc);
    fprintf( stderr, "%s\n",ldap_err2string (ldap_get_errno(ld1)) );
    exit(5);
}

/* check result of the bind */
rc = ldap_result (ld1, msg, 1, NULL, &result);
if (rc == -1 || rc == 0)
{
    fprintf( stderr, "ERROR\n");
    fprintf( stderr, "ldap_result after ldap_simple_bind failed rc=%d\n",rc);
    fprintf( stderr, "%s\n",ldap_err2string (ldap_get_errno(ld1)) );
    exit(5);
}

/* parse result of the bind pulling out the control responses */ /*TRUE*/
controls = 0;
parserc = ldap_parse_result (ld1, result, &rc, 0, 0, 0, &controls, 1);
if (rc != LDAP_SUCCESS)
{
    fprintf( stderr, "%s\n",ldap_err2string(rc) );
    exit(rc);
}

```

Figure B-4 Code sample

```

/* check the password policy responses in the control */
rc = ldap_parse_pwdpolicy_response(controls,
                                   &ctrlerr,
                                   &ctrlwarn,
                                   &ctrlres);
if ( ! (rc == LDAP_SUCCESS || rc == LDAP_CONTROL_NOT_FOUND) )
{
    fprintf( stderr, "ERROR\n");
    fprintf( stderr, "ldap_parse_pwdpolicy_response failed rc=%d\n",rc);
    fprintf( stderr, "%s\n",ldap_err2string(rc) );
    exit(5);
}

/* check the control response, changeAfterReset is 5 */
if ( ctrlerr == LDAP_CHANGE_AFTER_RESET ){

    /* tell the user their password has expired and to enter a new password */
    strcpy(newPW, getpass("password expired, enter new password "));
    strcpy(verPW, getpass("re-enter new password for verification "));

    /* if they entered a new password successfully
    then change their password using the modify delete-add format
    (this is the only format allowed at this point) */

    if ( strcmp(newPW,verPW) == 0) {

        strcpy(delVals[0],pw);
        strcpy(delVals[1],NULL);
        delete_mod.mod_op = LDAP_MOD_DELETE;
        delete_mod.mod_type = "userpassword";
        delete_mod.mod_vals.modv_strvals = (char **)malloc(18);
        delete_mod.mod_vals.modv_strvals[0] = *delVals;
        delete_mod.mod_vals.modv_strvals[1] = NULL;

        strcpy(addVals[0],newPW);
        strcpy(addVals[1],NULL);
        add_mod.mod_op = LDAP_MOD_ADD;
        add_mod.mod_type = "userpassword";
        add_mod.mod_vals.modv_strvals = (char **)malloc(18);
        add_mod.mod_vals.modv_strvals[0] = *addVals;
        add_mod.mod_vals.modv_strvals[1] = NULL;

        pmods[0]= &delete_mod;
        pmods[1]= &add_mod;
        pmods[2]= NULL;

        servercontrol = (LDAPControl **)malloc( 2, sizeof(LDAPControl *));
        servercontrol[0] = &pwdPolicyCtl;
        servercontrol[1] = NULL;
    }
}

```

Figure B-5 Code sample

```

rc=0;
    rc = ldap_modify_ext (ld1,
                        bindDn,
                        pmods,
                        NULL,
                        NULL,
                        &msgid);

    rc = ldap_result (ld1, msg, 1, NULL, &result);
    if (rc == -1 || rc == 0)
    {
        fprintf( stderr, "ERROR\n");
        fprintf( stderr, "ldap_result after ldap_modify_ext failed rc=%d\n",rc);
        fprintf( stderr, "%s\n",ldap_err2string (ldap_get_errno(ld1)) );
        exit(5);
    }

    /* parse result of the modify pulling out the control responses */ /*TRUE*/
    controls = 0;
    parserc = ldap_parse_result (ld1, result, &rc, 0, 0, 0, &controls, 1);
    if (rc != LDAP_SUCCESS)
    {
        fprintf( stderr, "ERROR\n");
        fprintf( stderr, "ldap_parse_result after ldap_modify_ext failed rc=%d\n",rc);
        fprintf( stderr, "%s\n",ldap_err2string(rc) );
        exit(5);
    }

    rc = ldap_parse_pwdpolicy_response(controls,
                                       &ctrlerr,
                                       &ctrlwarn,
                                       &ctrlres);
    if ( ! (rc == LDAP_SUCCESS || rc == LDAP_CONTROL_NOT_FOUND) )
    {
        fprintf( stderr, "%s\n",ldap_err2string(rc) );
        exit(rc);
    }
}

```

Figure B-6 Code sample

```
        fprintf( stderr, "password successfully changed\n");
    }
    else {
bindDn);
        fprintf( stderr, "new password verification failed, login failed for bindDn %s\n",
                bindDn);
        exit(5);

        } /* verify new password*/
    } /* if changeAfterReset */

    fprintf( stderr, "login successful for bindDN %s\n", bindDn);

    /* continue on to do other processing here, including any cleanup */
} /* main */
```

Figure B-7 Code sample

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 314. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *I/O Configuration Using z/OS HCD and HCM*, SG24-7804
- ▶ *Implementing PKI Services on z/OS*, SG24-6968
- ▶ *Understanding LDAP - Design and Implementation*, SG24-4986-01

Other publications

These publications are also relevant as further information sources:

- ▶ *Hardware Configuration Definition User's Guide*, SC33-7988
- ▶ *IBM Tivoli Directory Server Client Programming for z/OS V1R12.0*, SA23-2214-04
- ▶ *V1R10.0 IBM Tivoli Directory Server Plug-in Reference for z/OS*, SA76-0148-00
- ▶ *z/OS MVS Planning: Workload Management*, SA22-7602
- ▶ *z/OS UNIX System Services Planning*, GA22-7800
- ▶ *z/OS V1R12.0 IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191-05
- ▶ *z/OS V1R12.0 Network Authentication Service Administration*, SC24-5926
- ▶ *z/OS V1R12.0 Security Server RACF Command Language Reference*, SA22-7687
- ▶ *z/OS V1R12.0 Security Server RACF Security Administrator's Guide*, SA22-7683
- ▶ *z/OS V1R12.0 Security Server RACF System Programmer's Guide*, SA22-7861
- ▶ *z/OS V1R12.0 System SSL Programming*, SC24-5901

Online resources

These Web sites are also relevant as further information sources:

- ▶ IBM Tivoli Directory Server website
<http://www-01.ibm.com/software/tivoli/products/directory-server/>

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

A

- Access Control List (ACL) 8
- ACL see *Access Control List (ACL)*
- aclEntry 110
- aclPropagate 112
- activity log 8
- Advanced Replication 31
- Aliases 31
- Attribute Encryption 31
- attrOverflowSize configuration value 39
- AUDIT operator modify command 285
- authentication
 - anonymous or unauthenticated access 93
 - example 93
 - CRAM-MD5 bind
 - example 95, 97
 - DIGEST-MD5 bind 96
 - Kerberos 98
 - operations 92
 - SASL EXTERNAL (SSL) 101
 - certificate mapping 101
 - certificate mapping example 102
 - enabling 101
 - simple bind 93
 - example 94
 - supported bind or authentication mechanisms 92

B

- backend
 - CDBM 30
 - described 30
 - EXOP 30
 - features list 30
 - GDBM 30
 - LDBM 30
 - overview 30
 - Schema 30
 - SDBM 30
 - TDBM 30
- backend data stores
 - CDBM 6
 - GDBM 6
 - LDBM 6
 - Plug-in 6
 - Schema 6
 - SDBM 6
 - simple description 6
 - TDBM 6
- Basic Replication 31
- Bulk Load 31
- Bulk Unload 31

C

- CDBM backend 30
 - Advantages 44
 - configuration 44, 241
 - automated configuration using dsconfig utility 45
 - example 241
 - manual configuration 45
 - data store 6
 - detailed description 44
 - Sysplex 225
 - tuning 46
 - usage examples
 - display the cn=Replication,cn=configuration entry 46
 - modify the ibm-replicationOnHold attribute in the cn=Replication,cn=configuration entry 46
- change logging 32
- configuration
 - enable SSL authentication 198
 - GDBM backend 185
 - enable GDBM for a LDBM based backend 185
 - enable GDBM for a TDBM backend 187
 - LDBM backend 176
 - prerequisite steps 176
 - setup using dsconfig 176
 - start and verify server 178
 - password policy example 210
 - schema
 - loading the IBM supplied schema 192
 - using the sample.ldif file 193
 - SDBM backend 191
 - securing the administrator id 194
 - TDBM backend 178
 - prerequisites 179
 - setting up DB2 179
 - setting up using dsconfig 181
 - starting and verifying the server 184
 - using CRAM-MD5 and DIGEST-MD5 binds 196
- configuration examples
 - advanced replication 240
 - configure a CDBM backend 241
 - master-replica 245
 - topologies 240
 - replication 229–230
 - master-replica 230
 - master-replica configuration parameters 230
 - master-replica configuration steps 230
 - peer to peer 234
 - peer to peer configuration parameters 235
 - peer to peer configuration steps 235
- CRAM-MD5 95, 97
 - binds 196
- cross-system coupling facility (XCF) 9
 - sharing group 219
- CTRACE 289

D

- DB2 21
 - buffer pools 38
 - tuning 37
- debugging
 - CTRACE 289
 - captured output 289
 - direct debug output example 290
 - directing debug output to 290
 - in-memory trace table 290
 - debug facility output 288
 - debug level examples 289
 - dsconfig 288
 - trace output 288
 - dynamic debugging command examples 289
 - how to debug problems 288
 - output on the screen 288
 - trace file 288
 - overview 288
 - server debug mode 288
 - debug level 289
 - setting 289
 - setting -d parameter 289
 - setting dynamic debugging 289
 - setting LDAP_DEBUG environment variable 289
- DIGEST-MD5 96
 - binds 196
- DIR_ENTRY table 38
- DIR_SEARCH table 39
- directory information tree (DIT) 5
 - example 5
- distinguished name (DN) 93
 - workload classification by 170
- DIT see *directory information tree (DIT)*
- DN see *distinguished name (DN)*
- ds.conf 7, 33
- ds.envvars 7
- dsconfig utility
 - debugging 288
 - trace output 288
 - description 23
 - generated files 25
 - input files 23–24
 - usage overview 24

E

- EXOP
 - backend 30

F

- Forwarding (cascading) replication 12
 - forwarding topology 12
 - sample Forwarding topology 12
- forwarding replication 6

G

- Gateway replication 15
 - gateway server 15

- sample gateway topology 15
- GDBM backend 30
 - Advantages 66
 - configuration 66, 185
 - automated configuration
 - DB2-based GDBM configuration 66
 - file-based GDBM 66
 - automated configuration using dsconfig utility 66
 - change logging
 - enabling 68
 - Additional RACF configuration 68
 - LDBM backend 185
 - manual configuration 67
 - DB2-based GDBM 67
 - file-based GDBM 67
 - TDBM backend 187
 - data store 6
 - detailed description 64
 - Sysplex 223
 - tuning 71
 - usage examples 70
- GetEffectiveAcl 117

H

- HCD plug-in 256
 - description 256
 - function 256
 - HCD schema file 264
 - LDAP server configuration requirements 258
 - loading the HCD schema 264
 - setup steps 259
 - structure 256
 - usage 264
 - authentication 264
 - example 265
- HCD schema file 264

I

- IBM TDS for z/OS LDAP features 31
 - Advanced Replication 31
 - Aliases 31
 - Attribute Encryption 31
 - Basic Replication 31
 - Bulk Load 31
 - Bulk Unload 31
 - change logging 32
 - Multi-Server Operational Modes 32
 - Native Authentication 32
 - Password Policy 32
 - Policy Director 32
 - RACF Administration 32
 - Referrals 32
- IBM Tivoli Directory Server (TDS) for z/OS 256
 - advanced replication 6
 - advanced replication topologies 6
 - forwarding replication 6
 - gateway replication 6

- master-replica replication 6
 - peer-to-peer replication 6
- directory 4
- directory architecture 5
 - attribute value 5
 - attributes 5
 - directory information tree (DIT) 5
 - directory information tree example 5
 - distinguished name (DN) 5
 - entries 5
 - relative distinguished name (RDN) 5
 - type 5
- features 4
- operational modes 5
 - multiple single-server mode 5
 - multi-server mode 5
 - single-server mode 5
- Planning 20–21
 - activity and audit logging 23
 - advanced replication 23
 - configuring using the dsconfig utility 23
 - DB2 and ODBC 21
 - Integrated Cryptographic Security Facility (ICSF) 21
 - LDAP password policy 23
 - planning considerations before using the dsconfig utility 22
 - Resource Access Control Facility (RACF) 21
 - storing user passwords for LDBM or TDBM backend 22
 - supported backends 20
 - where and how to store user passwords 22
 - Workload Manager (WLM) 21
 - z/OS Cryptographic Services System SSL 21
 - z/OS Integrated Security Services Network Authentication Service 21
 - z/OS UNIX System Services file system 21
 - Version 3 LDAP client and server 4
- ICSF see *Integrated Cryptographic Security Facility (ICSF)*
- Integrated Cryptographic Security Facility (ICSF) 21

L

- LDAP access security 8
- LDAP client 7
- LDAP HCD support
 - functions 257
 - security 256
 - setup 257
- LDAP password policy 23
- LDAP server
 - as daemon 7
 - configuration requirements 258
 - HCD plug-in 258
 - SMF records 8
 - statistics 268
- LDAP_DEBUG environment variable 289
- LDBM backend 30
 - Advantages 39
 - configuration 40, 176

- automated configuration using dsconfig utility 40
 - manual configuration 40
 - sample LDBM backend-specific section of the ds.conf configuration file 40
 - porting existing LDBM database 41
 - shipped example for starting LDBM backend 41
- data store 6
- detailed description 39
- storing user passwords 22
- Sysplex 218
- tuning 41
 - database commit processing 43
 - memory considerations 42
 - sample benchmark data 43
 - space required 43
 - startup time 42

M

- master-server replication
 - description 10
 - read-only replica server 10
 - replicated data 10
 - replication context 10
 - sample Master-Replica topology 11
- monitoring
 - activity log 282
 - configuration 283
 - configuration archiving 283
 - configuration logfile option 283
 - features 283
 - advanced replication 276
 - by use of ldapsearch 276
 - example 276
 - extended operations 279
 - extended operations description 280
 - monitor the advanced replication processing 280
 - objects returned 278
 - retrieve replication topology information 277
 - retrieving operational attributes for the replication agreement 281
 - retrieving operational attributes for the replication context 280
 - retrieving operational attributes of ibm-replication-Context 280
 - specific configured suffix or specific replication context 277
 - subtree scope search 276
- audit logging 285
 - audit levels 285
 - AUDIT operator modify command 285
 - audit parameter 285
- operations monitor 284
 - configuration option 284
- server monitoring 268
 - distinguished name entries 268
 - example 270
 - LDAP server statistics returned 268
 - scope 268
 - scope=sub examples 268
 - search base 268

- multiple single-server mode 5
- multiple single-server operational mode 7
- multi-server mode 5, 8
 - intended use 9
 - shared UNIX System Services based backend files 10
 - TDBM backend caches 10
 - use of cross-system coupling facility (XCF) 9
- Multi-Server Operational Modes 32

N

- Native Authentication 32

O

- objectclasses attribute value 82
- ODBC 21
- Operations Monitor 171
- ownerPropagate 112

P

- Password Policy 32
 - example 210
- peer-to-peer replication 13
 - sample Peer-to-Peer topology 14
- Persistent Search 136
- plug-ins
 - building 162
 - client-operation 160
 - request message types 160
 - sample plug-in 163
 - description 156
 - exploiters 166
 - extending functions 8
 - post-operation 158
 - pre-operation 158
 - request message types 159
 - server flow for the request processing illustrated 157
 - server processing 156
- Policy Director 32

R

- RACF see *Resource Access Control Facility (RACF)*
- RDN see *relative distinguished name (RDN)*
- Redbooks Web site 314
 - Contact us xiii
- Referrals 32
- relative distinguished name (RDN) 5
- reliability, availability, and scalability 140
- remote security services 16
- REORG 37
- replication 141
 - configuration
 - consumer 146
 - maintain an active replication topology 150
 - maintenance mode 145
 - partially replicate 154
 - requisite changes 144
 - scheduled replication 153

- server ID 145
- SSL 145
- supplier 146
- synchronization among all the servers 149
- Sysplex and replication 143
- terminology 141
- topology 142
 - forwarding/cascading 143
 - gateway 143
 - master-replica 143
 - peer-peer 143
 - Sysplex and replication 144
- Resource Access Control Facility (RACF) 21
 - Administration 32
- RUNSTATS 37

S

- sample.Idif file 193
- SASL EXTERNAL (SSL) 101
- schema
 - attribute type 74–75
 - attribute syntaxes 75
 - matching rules 77
 - matching rules example 78
 - required attribute values when adding a new attribute 79
 - attribute values 74
 - attributes 74
 - configuration 74
 - schema.IBM.Idif 75
 - schema.user.Idif 75
 - define your own schema 82
 - example 83
 - description 74
 - object class 74, 81
 - objectclasses attribute value 82
 - supported object classes 81
 - RACF custom fields 87
- schema.IBM.Idif 75
- schema.user.Idif 75
- SDBM
 - backend 30
 - configuration 191
 - backend data store 6
- SDBM backend
 - Advantages 47
 - configuration 47
 - automated configuration using dsconfig utility 47
 - manual configuration 48
 - sample SDBM backend-specific section of the ds.conf configuration file 48
 - usage examples 50
 - add a new resource profile 51
 - modifying existing RACF user 50
 - refresh the RACF FACILITY class 52
 - search 51
 - detailed description 46
 - directory hierarchy 49
 - RACF General Resources 55
 - usage examples 55

- search 52
 - supported search filters 53
 - tuning 54
- Secure Socket Layer (SSL) 133
- security
 - access control lists 108
 - access class for the authorization related attributes 109
 - aclEntry 110
 - aclPropagate 112
 - attribute classes 108
 - attribute level action 108
 - dynamic group 121
 - entryOwner 110
 - filtered access control 116
 - GetEffectiveAcl 117
 - groups 121
 - nested group 123
 - normalization rules 111
 - ownerPropagate 112
 - permissions 113
 - permissions calculation 115
 - precedence 114
 - propagating ACLs and entry ownership 112
 - querying group membership 123
 - static group 121
 - subject seeking authorization 114
 - authentication 91
 - authorization and audit 92
 - encryption and hashing 132
 - one-way hashing algorithms 132
 - pwEncryption option 132
 - secretEncryption option 132
 - two-way encryption algorithms 132
 - group gathering 126
 - native authentication 104
 - changing or updating a user's password or password phrase 107
 - configuring 107
 - overview 90
 - password policy 127
 - global password policy 127
 - group password policy 127
 - individual password policy 128
 - native authentication and expired passwords 131
 - password policy attributes 128
 - password policy operational attributes 130
 - when is it checked? 130
- Secure Socket Layer (SSL) and Transport Layer Security (TLS) 133
- security concepts 90
 - audit 90
 - authentication 90
 - authorization 90
 - confidentiality 90
 - mapping of general IT security concepts to IBM TDS for z/OS features 90
- SSL (Secure Socket Layer) and TLS (Transport Layer Security)
 - certificate key repositories 134

- configuration 135
- serverSysplexGroup 218
- single-server architecture 6
- single-server mode 5
- SMF records 8
- SSL see *Secure Socket Layer (SSL)*
- Sysplex 140
 - CDBM 225
 - GDBM 223
 - LDBM 218
 - using the same configuration 218
 - multiserver 218
 - serverSysplexGroup 218
 - TDBM 220
 - options in the configuration file 220
 - using the same configuration 220
 - XCF sharing group 219

T

- Tablespace LOCKSIZE 38
- TDBM
 - Sysplex 220
- TDBM backend 30
 - advantages 32
 - automated configuration using dsconfig 33
 - generated members 33
 - configuration 33, 178
 - data store 6
 - detailed description 32
 - manual configuration 33
 - sample TDBM backend-specific section of the ds.conf configuration file 33
 - porting existing databases 33–34
 - porting ISS TDBM database 34
 - procedure 34
 - storing user passwords 22
 - tuning 37
 - attrOverflowSize configuration value 39
 - cache tuning 37
 - configuration file 39
 - DB2 buffer pools 38
 - DB2 tuning 37
 - maintain database statistics (RUNSTATS) 37
 - reorganize the database (REORG) 37
 - Size of DIR_ENTRY table's DN_TRUNC column 38
 - Size of DIR_SEARCH table's VALUE column 39
 - Tablespace LOCKSIZE 38
 - TDBM SQL 38
 - usage examples
 - add a directory entry 35
 - delete an entry 36
 - load all of the entries from an input LDIF file 36
 - modify a directory entry 35
 - search for an entry 35
 - unload all entries 36
 - when to use 32
- TDBM SQL 38
- TDS see *IBM Tivoli Directory Server (TDS) for z/OS*
- TLS see *Transport Layer Security (TLS)*

Transport Layer Security (TLS) 133

W

WLM see *Workload Manager (WLM)*

Workload Manager (WLM) 21

support 9

load balancing 9

X

XCF see *cross-system coupling facility (XCF)*

Z

z/OS Cryptographic Services System SSL 21

z/OS Integrated Security Services Network Authentication Service 21

z/OS UNIX System Services file system 21

z/OS Workload Manager 168

configuration options 169

classification rules 169

incoming requests 169

service classes 169

workload classification by requestor distinguished name (DN) 170

workload classification by requestor IP address 170

enclave 168

execution velocity goals 168

goal setting 168

health service 171

search pattern 168

service class 168

using with Operations Monitor 171



Redbooks

IBM Tivoli Directory Server for z/OS

(0.5" spine)
0.475" x 0.873"
250 <-> 459 pages



IBM Tivoli Directory Server for z/OS



Redbooks®

Technical overview of Tivoli Directory Server

Concepts, planning, and configuration examples

Basic and advanced replication

This IBM Redbooks publication examines the IBM Tivoli Directory Server for z/OS. IBM Tivoli Directory Server is a powerful Lightweight Directory Access Protocol (LDAP) infrastructure that provides a foundation for deploying comprehensive identity management applications and advanced software architectures.

This publication provides an introduction to the IBM Tivoli Directory Server for z/OS that provides a brief summary of its features and an examination of the possible deployment topologies. It discusses planning a deployment of IBM Tivoli Directory Server for z/OS, which includes prerequisites, planning considerations, and data stores, and provides a brief overview of the configuration process. Additional chapters provide a detailed discussion of the IBM Tivoli Directory Server for z/OS architecture that examines the supported back ends, discusses in what scenarios they are best used, and provides usage examples for each back end. The discussion of schemas breaks down the schema and provides guidance on extending it. A broad discussion of authentication, authorization, and security examines the various access protections, bind mechanisms, and transport security available with IBM Tivoli Directory Server for z/OS. This chapter also provides an examination of the new Password Policy feature. Basic and advanced replication topologies are also covered. A discussion on plug-ins provides details on the various types of plug-ins, the plug-in architecture, and creating a plug-in, and provides an example plug-in. Integration of IBM Tivoli Directory Server for z/OS into the IBM Workload Manager environment is also covered.

This publication also provides detailed information about the configuration of IBM Tivoli Directory Server for z/OS. It discusses deploying IBM Tivoli Directory Server for z/OS on a single system, with examples of configuring the available back ends. Configuration examples are also provided for deploying the server in a Sysplex, and for both basic and advanced replication topologies. Finally it provides guidance on monitoring and debugging IBM Tivoli Directory Server for z/OS.

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

SG24-7849-00

ISBN 0738435724