Sun Java System Directory Server Enterprise Edition 6.3 Reference

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–2765
April 2008

Contents

Figures

Tables

Examples

Preface

This book describes product architecture, configuration, tools, APIs, and schema for Directory Server and Directory Proxy Server.

Who Should Use This Book

This *Reference* is intended for directory service administrators, designers, and developers.

Before You Read This Book

Review pertinent information in the *Sun Java System Directory Server Enterprise Edition 6.3 Release Notes*.

If you are deploying Directory Server Enterprise Edition software in production, also review pertinent information in the *Sun Java System Directory Server Enterprise Edition 6.3 Deployment Planning Guide*.

How This Book Is Organized

Part I covers Directory Server features and architecture.

Part II covers Directory Proxy Server features and architecture.

Appendix A, "Directory Server Resource Kit File Reference" describes the installed product layout for Directory Server Resource Kit.

Directory Server Enterprise Edition Documentation Set

This Directory Server Enterprise Edition documentation set explains how to use Sun Java System Directory Server Enterprise Edition to evaluate, design, deploy, and administer directory services. In addition, it shows how to develop client applications for Directory Server Enterprise Edition. The Directory Server Enterprise Edition documentation set is available at `http://docs.sun.com/coll/1224.4`.

For an introduction to Directory Server Enterprise Edition, review the following documents in the order in which they are listed.

TABLE P–1 Directory Server Enterprise Edition Documentation

Document Title	Contents
Sun Java System Directory Server Enterprise Edition 6.3 Release Notes	Contains the latest information about Directory Server Enterprise Edition, including known problems.
Sun Java System Directory Server Enterprise Edition 6.3 Documentation Center	Contains links to key areas of the documentation set.
Sun Java System Directory Server Enterprise Edition 6.3 Evaluation Guide	Introduces the key features of this release. Demonstrates how these features work and what they offer in the context of a fictional deployment that you can implement on a single system.
Sun Java System Directory Server Enterprise Edition 6.3 Deployment Planning Guide	Explains how to plan and design highly available, highly scalable directory services based on Directory Server Enterprise Edition. Presents the basic concepts and principles of deployment planning and design. Discusses the solution life cycle, and provides high-level examples and strategies to use when planning solutions based on Directory Server Enterprise Edition.
Sun Java System Directory Server Enterprise Edition 6.3 Installation Guide	Explains how to install the Directory Server Enterprise Edition software. Shows how to select which components to install, configure those components after installation, and verify that the configured components function properly. For instructions on installing Directory Editor, go to `http://docs.sun.com/coll/DirEdit_05q1`. Make sure you read the information in *Sun Java System Directory Server Enterprise Edition 6.3 Release Notes* concerning Directory Editor before you install Directory Editor.
Sun Java System Directory Server Enterprise Edition 6.3 Migration Guide	Provides migration instructions from the earlier versions of Directory Server, Directory Proxy Server, and Identity Synchronization for Windows.

TABLE P–1 Directory Server Enterprise Edition Documentation *(Continued)*

Document Title	Contents
Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide	Provides command-line instructions for administering Directory Server Enterprise Edition. For hints and instructions on using the Directory Service Control Center, DSCC, to administer Directory Server Enterprise Edition, see the online help provided in DSCC. For instructions on administering Directory Editor, go to `http://docs.sun.com/coll/DirEdit_05q1`. For instructions on installing and configuring Identity Synchronization for Windows, see Part II, Installing Identity Synchronization for Windows.
Sun Java System Directory Server Enterprise Edition 6.3 Developer's Guide	Shows how to develop directory client applications with the tools and APIs that are provided as part of Directory Server Enterprise Edition.
Sun Java System Directory Server Enterprise Edition 6.3 Reference	Introduces the technical and conceptual foundations of Directory Server Enterprise Edition. Describes its components, architecture, processes, and features. Also provides a reference to the developer APIs.
Sun Java System Directory Server Enterprise Edition 6.3 Man Page Reference	Describes the command-line tools, schema objects, and other public interfaces that are available through Directory Server Enterprise Edition. Individual sections of this document can be installed as online manual pages.
Sun Java System Directory Server Enterprise Edition 6.3 Troubleshooting Guide	Provides information for defining the scope of the problem, gathering data, and troubleshooting the problem areas using various tools.
Sun Java System Identity Synchronization for Windows 6.0 Deployment Planning Guide	Provides general guidelines and best practices for planning and deploying Identity Synchronization for Windows

Related Reading

The SLAMD Distributed Load Generation Engine is a Java™ application that is designed to stress test and analyze the performance of network-based applications. It was originally developed by Sun Microsystems, Inc. to benchmark and analyze the performance of LDAP directory servers. SLAMD is available as an open source application under the Sun Public License, an OSI-approved open source license. To obtain information about SLAMD, go to `http://www.slamd.com/`. SLAMD is also available as a java.net project. See `https://slamd.dev.java.net/`.

Java Naming and Directory Interface (JNDI) technology supports accessing the Directory Server using LDAP and DSML v2 from Java applications. For information about JNDI, see `http://java.sun.com/products/jndi/`. The *JNDI Tutorial* contains detailed descriptions and examples of how to use JNDI. This tutorial is at `http://java.sun.com/products/jndi/tutorial/`.

Directory Server Enterprise Edition can be licensed as a standalone product, as a component of Sun Java Enterprise System, as part of a suite of Sun products, such as the Sun Java Identity Management Suite, or as an add-on package to other software products from Sun. Java Enterprise System is a software infrastructure that supports enterprise applications distributed across a network or Internet environment. If Directory Server Enterprise Edition was licensed as a component of Java Enterprise System, you should be familiar with the system documentation at `http://docs.sun.com/coll/1286.3`.

Identity Synchronization for Windows uses Message Queue with a restricted license. Message Queue documentation is available at `http://docs.sun.com/coll/1307.2`.

Identity Synchronization for Windows works with Microsoft Windows password policies.

- Information about password policies for Windows 2003 is available in the Microsoft documentation online.
- Information about the Microsoft Certificate Services Enterprise Root certificate authority is available in the Microsoft support documentation online.
- Information about configuring LDAP over SSL on Microsoft systems is available in the Microsoft support documentation online.

Redistributable Files

Directory Server Enterprise Edition does not provide any files that you can redistribute.

Default Paths and Command Locations

This section explains the default paths used in the documentation, and gives the locations of commands on different operating systems and deployment types.

Default Paths

The table in this section describes the default paths that are used in this document. For complete descriptions of the files installed, see the following product documentation.

- Chapter 14, "Directory Server File Reference,"
- Chapter 25, "Directory Proxy Server File Reference,"
- Appendix A, "Directory Server Resource Kit File Reference"

TABLE P–2 Default Paths

Placeholder	Description	Default Value
install-path	Represents the base installation directory for Directory Server Enterprise Edition software. The software is installed in directories below this base *install-path*. For example, Directory Server software is installed in *install-path*`/ds6/`.	When you install from a zip distribution using `dsee_deploy`(1M), the default *install-path* is the current directory. You can set the *install-path* using the `-i` option of the `dsee_deploy` command. When you install from a native package distribution, such as you would using the Java Enterprise System installer, the default *install-path* is one of the following locations: ■ Solaris systems - `/opt/SUNWdsee/`. ■ Red Hat systems - `/opt/sun/`. ■ Windows systems - `C:\Program Files\Sun\JavaES5\DSEE`.
instance-path	Represents the full path to an instance of Directory Server or Directory Proxy Server. The documentation uses `/local/ds/` for Directory Server and `/local/dps/` for Directory Proxy Server.	No default path exists. Instance paths must nevertheless always be found on a *local* file system. The following directories are recommended: `/var` on Solaris systems `/global` if you are using Sun Cluster
serverroot	Represents the parent directory of the Identity Synchronization for Windows installation location	Depends on your installation. Note the concept of a *serverroot* no longer exists for Directory Server.
`isw-`*hostname*	Represents the Identity Synchronization for Windows instance directory	Depends on your installation
`/`*path*`/`*to*`/cert8.db`	Represents the default path and file name of the client's certificate database for Identity Synchronization for Windows	*current-working-dir*`/cert8.db`
serverroot`/isw-`*hostname*`/logs/`	Represents the default path to the Identity Synchronization for Windows local logs for the System Manager, each connector, and the Central Logger	Depends on your installation
serverroot`/isw-`*hostname*`/logs/central/`	Represents the default path to the Identity Synchronization for Windows central logs	Depends on your installation

Command Locations

The table in this section provides locations for commands that are used in Directory Server Enterprise Edition documentation. To learn more about each of the commands, see the relevant man pages.

TABLE P–3 Command Locations

Command	Java ES, Native Package Distribution	Zip Distribution
cacaoadm	Solaris - /usr/sbin/cacaoadm	Solaris - *install-path*/dsee6/cacao_2/usr/sbin/cacaoadm
	Red Hat - /opt/sun/cacao/bin/cacaoadm	Red Hat, HP-UX - *install-path*/dsee6/cacao_2/cacao/bin/cacaoadm
	Windows - *install-path*\share\cacao_2\bin\cacaoadm.bat	Windows - *install-path*\dsee6\cacao_2\bin\cacaoadm.bat
certutil	Solaris - /usr/sfw/bin/certutil	*install-path*/dsee6/bin/certutil
	Red Hat - /opt/sun/private/bin/certutil	
dpadm(1M)	*install-path*/dps6/bin/dpadm	*install-path*/dps6/bin/dpadm
dpconf(1M)	*install-path*/dps6/bin/dpconf	*install-path*/dps6/bin/dpconf
dsadm(1M)	*install-path*/ds6/bin/dsadm	*install-path*/ds6/bin/dsadm
dsccmon(1M)	*install-path*/dscc6/bin/dsccmon	*install-path*/dscc6/bin/dsccmon
dsccreg(1M)	*install-path*/dscc6/bin/dsccreg	*install-path*/dscc6/bin/dsccreg
dsccsetup(1M)	*install-path*/dscc6/bin/dsccsetup	*install-path*/dscc6/bin/dsccsetup
dsconf(1M)	*install-path*/ds6/bin/dsconf	*install-path*/ds6/bin/dsconf
dsee_deploy(1M)	Not provided	*install-path*/dsee6/bin/dsee_deploy
dsmig(1M)	*install-path*/ds6/bin/dsmig	*install-path*/ds6/bin/dsmig
entrycmp(1)	*install-path*/ds6/bin/entrycmp	*install-path*/ds6/bin/entrycmp
fildif(1)	*install-path*/ds6/bin/fildif	*install-path*/ds6/bin/fildif
idsktune(1M)	Not provided	At the root of the unzipped zip distribution

TABLE P–3 Command Locations *(Continued)*

Command	Java ES, Native Package Distribution	Zip Distribution
`insync`(1)	*install-path*`/ds6/bin/insync`	*install-path*`/ds6/bin/insync`
`ns-accountstatus`(1M)	*install-path*`/ds6/bin/ns-accountstatus`	*install-path*`/ds6/bin/ns-accountstatus`
`ns-activate`(1M)	*install-path*`/ds6/bin/ns-activate`	*install-path*`/ds6/bin/ns-activate`
`ns-inactivate`(1M)	*install-path*`/ds6/bin/ns-inactivate`	*install-path*`/ds6/bin/ns-inactivate`
`repldisc`(1)	*install-path*`/ds6/bin/repldisc`	*install-path*`/ds6/bin/repldisc`
`schema_push`(1M)	*install-path*`/ds6/bin/schema_push`	*install-path*`/ds6/bin/schema_push`
`smcwebserver`	Solaris, Linux - `/usr/sbin/smcwebserver` Windows - *install-path*`\share\webconsole\bin\smcwebserver`	This command pertains only to DSCC when it is installed using native packages distribution.
`wcadmin`	Solaris, Linux - `/usr/sbin/wcadmin` Windows - *install-path*`\share\webconsole\bin\wcadmin`	This command pertains only to DSCC when it is installed using native packages distribution.

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–4 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`**
		`Password:`

AaBbCc123	A placeholder to be replaced with a real name or value	The command to remove a file is `rm` *filename*.

TABLE P–4 Typographic Conventions *(Continued)*

Typeface	Meaning	Example
AaBbCc123	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file.

Shell Prompts in Command Examples

The following table shows default system prompts and superuser prompts.

TABLE P–5 Shell Prompts

Shell	Prompt
C shell on UNIX and Linux systems	`machine_name%`
C shell superuser on UNIX and Linux systems	`machine_name#`
Bourne shell and Korn shell on UNIX and Linux systems	`$`
Bourne shell and Korn shell superuser on UNIX and Linux systems	`#`
Microsoft Windows command line	`C:\`

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P–6 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.

TABLE P–6 Symbol Conventions *(Continued)*

Symbol	Description	Example	Meaning
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Searching Sun Product Documentation

Besides searching for Sun product documentation from the docs.sun.com web site, you can use a search engine of your choice by typing the following syntax in the search field:

search-term `site:docs.sun.com`

For example, to search for `Directory Server`, type the following:

```
"Directory Server" site:docs.sun.com
```

To include other Sun web sites in your search, such as java.sun.com, www.sun.com, and developers.sun.com, use `sun.com` in place of `docs.sun.com` in the search field.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to `http://docs.sun.com` and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 820-2765.

PART I

Directory Server Reference

This part explains how Directory Server works. The information here is primarily descriptive. For instructions, try Part I, “Directory Server Administration,” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide* instead.

This part covers the following chapters.

- Chapter 1, “Directory Server Overview,”
- Chapter 2, “Directory Server Security,”
- Chapter 3, “Directory Server Monitoring,”
- Chapter 4, “Directory Server Replication,”
- Chapter 5, “Directory Server Data Caching,”
- Chapter 6, “Directory Server Indexing,”
- Chapter 7, “Directory Server Logging,”
- Chapter 8, “Directory Server Groups and Roles,”
- Chapter 9, “Directory Server Class of Service,”
- Chapter 10, “Directory Server DSMLv2,”
- Chapter 11, “Directory Server Internationalization Support,”
- Chapter 12, “Directory Server LDAP URLs,”
- Chapter 13, “Directory Server LDIF and Search Filters,”
- Chapter 14, “Directory Server File Reference,”

For additional reference information, see *Sun Java System Directory Server Enterprise Edition 6.3 Man Page Reference*.

CHAPTER 1

Directory Server Overview

This chapter outlines the architecture of Directory Server. This chapter includes the following topics.

- "Introduction to Directory Server" on page 33
- "Directory Server Architecture" on page 34

Introduction to Directory Server

Directory Server serves directory data to standards compliant LDAP and DSML applications. Directory Server stores the data in customized, binary tree databases, allowing quick searches even for large data sets.

Directories are object oriented databases. Directories organize their data objects, called entries, into a directory information tree, often called a DIT. Each entry is identified by a distinguished name, such as `uid=bjensen,ou=people,dc=example,dc=com`. The distinguished name identifies where the entry is located in the directory information tree. For example, `uid=bjensen,ou=people,dc=example,dc=com` is a user entry for Barbara Jensen on the `ou=people` branch of the `dc=example,dc=com` part of the tree.

Each directory entry has attributes. For entries that concern people, these attributes may reflect names, phone numbers, and email addresses, for example. An attribute has at least one type name, which is the name of the attribute. For example, people entries can have an attribute `surname`, which can also be called by the shorter name `sn`. Attributes can also have one or more values. For example, if Barbara Jensen marries Quentin Cubbins, and takes Quentin's surname, her entry could have `sn: Jensen` and `sn: Cubbins`.

Directories are designed to be fast when looking up entries based on the values of their attributes. An example query might be, "Find all the entries under `dc=example,dc=com` with surname Jensen." This fast lookup capability makes directories well suited for applications where you store information that must be read often. Directories are therefore good data stores for telephone and email information. Directories are also good for handling authentication credentials, identity information, and application configuration data.

Directory Server is also designed to handle high update rates as the information in the directory changes. Today, the size of many directory deployments mean that handling updates well can be as important as handling lookups.

Directory Server supports many directory related standards and RFCs. Directory Server allows fast data replication across the network for high availability. Directory Server lets you configure servers comprehensively without restarting them. Furthermore, Directory Server gives you extensive control over access to directory data.

The list of Directory Server features is too long to cover in a short introduction. *Sun Java System Directory Server Enterprise Edition 6.3 Evaluation Guide* includes a more extensive list. The other chapters in this part of this *Reference* help you to understand many of the features in detail.

Directory Server Architecture

This section succinctly addresses key concepts of Directory Server from the point of view of someone who must install and manage Directory Server. This section touches on the following topics.

- "Comparison of Software Installation and Server Instances" on page 34
- "Communication With Client Applications" on page 35
- "Directory Server Configuration" on page 36
- "Data Storage in Directory Server" on page 37
- "Data Replication Between Server Instances" on page 38
- "Access Control in Directory Server" on page 38

Comparison of Software Installation and Server Instances

For each installation of Directory Server software, you can create multiple server instances. Although you may create server instances in the place on the file system where you install the software, nothing requires you to put both the software and the instances side by side.

The Directory Server software you install includes the executable files, template data, and sample files needed to create, run, and manage actual servers. As the software is separate from the actual servers, you can apply patches or service packs to the software without changing the server data. You therefore do not need to patch each server instance, but instead only the software installation.

A Directory Server instance holds the configuration data and the directory data required to serve directory client applications. Although in production systems you carefully control the user identity of the server, you can typically create and run a Directory Server instance as any user on the system. The directory data belongs then to the user who created the instance.

In Chapter 14, "Directory Server File Reference," you see that "Software Layout for Directory Server" on page 247 is clearly separate from "Directory Server Instance Default Layout" on page 251. In particular, notice that the documentation mentions *install-path* when referring to the software installation, but *instance-path* when referring to a server instance.

Communication With Client Applications

Directory Server listens for LDAP and DSML client application traffic on the port numbers you configure. Directory Server listens for LDAP connections as soon as the server starts. Directory Server only listens for DSML connections over HTTP if you enable the DSML service.

By default, Directory Server listens for LDAP connections on port 389 if the instance was created by `root`, 1389 if the instance was created by non-`root`. By default, Directory Server listens for LDAP connections over SSL on port 636 if the instance was created by `root`, 1636 if the instance was created by non-`root`. The DSML/HTTP port number is not defined by default. Instead, you supply a port number when enabling the DSML service.

In order to enable client applications to reach Directory Server, you create instances on hosts with static IP addresses. The hostname is also usually referenced in DNS. Client applications typically need at least two pieces of information to access the directory.

1. The hostname, or at least the IP address, of the system on which Directory Server runs.
2. The port number on which Directory Server listens for client connections.

LDAP clients and servers do not usually open a new connection for every request. In the LDAP model, a client connects to the server to authenticate before performing other operations. The connection and authentication process is referred to as binding. Client applications can bind with credentials, but they can also bind anonymously. Directory Server lets you configure access accordingly both for known and anonymous clients. Client applications can also keep a connection open, but bind again, thus changing the authentication identity. This technique can reduce the costs of creating a new connection.

Once the bind has been performed and the client is authenticated, the client can request the following operations.

add	Add a new directory entry.
compare	Checks whether an attribute value is the same as a given value.
delete	Delete a directory entry.
modify	Change one or more attributes of a directory entry.

modDN Change the distinguished name of a directory entry.

This operation is for moving directory entries from one part of the directory information tree to another. For example, you could move `uid=bjensen,ou=employees,dc=example,dc=com` to `uid=bjensen,ou=people,dc=example,dc=com`.

When you move an parent entry, such as `ou=people,dc=example,dc=com`, the operation can take a very long time as Directory Server must move all child entries of the parent as well.

modRDN Change the relative distinguished name of a directory entry.

The relative distinguished name is the attribute value used to distinguish a directory entry from the others at the same level of the directory information tree.

This operation is for renaming directory entries. For example, you could rename `uid=bjensen,ou=employees,dc=example,dc=com` to `uid=bcubbins,ou=people,dc=example,dc=com`.

This operation is a special case of modDN. The modRDN operation is always relatively fast, however, as it involves modifying only leaf entries.

search Find all the directory entries under a specified point in the directory tree that have attribute values matching a filter.

A search filter can specify one or more attribute characteristics. For example, to find entries with the surname Jensen, you use the LDAP filter `(sn=Jensen)`. To find entries with surname Jensen and user ID beginning with the letter B, you use the LDAP filter `(&(sn=Jensen)(uid=b*))`.

When finished performing operations, a client can unbind. After unbinding, the connection is dropped by the client and the server. Client applications can also abandon operations, such as a search that is taking too long.

Directory Server can handle many client connections simultaneously. To handle connections, Directory Server consumes free file descriptors, and manages a number of threads. You can limit the system resources available to Directory Server through the server configuration. See Chapter 6, “Tuning System Characteristics and Hardware Sizing,” in *Sun Java System Directory Server Enterprise Edition 6.3 Deployment Planning Guide* for details.

Directory Server Configuration

Directory Server stores server instance configuration data in files, but the configuration data is also accessible over LDAP.

The files are stored under *instance-path* as follows. Directory Server stores the LDAP schema, which define what directory entries can contain, under *instance-path*`/config/schema/`. See *Sun Java System Directory Server Enterprise Edition 6.3 Man Page Reference* for reference information about the schema, and Chapter 12, "Directory Server Schema," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide* for instructions on managing schema. Directory Server stores other configuration information in the `dse.ldif`(4) file, *instance-path*`/config/dse.ldif`. Avoid updating this file by hand.

Over LDAP, the schema information is accessible under `cn=schema`. The other configuration information is accessible under `cn=config`. In practice, you do not generally update data under `cn=config` directly. Instead, you use either the web based Directory Service Control Center, or the `dsconf` command. Both Directory Service Control Center and the `dsconf` command change Directory Server over LDAP. Yet, both also spare you much of the complexity of making configuration adjustments with LDAP modify operations.

Almost all Directory Server product documentation is devoted to Directory Server configuration. In *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*, you find extensive instructions for accomplishing a variety of tasks using command line configuration tools. The Directory Service Control Center online help can help get you back on track when the Directory Service Control Center interface does not seem intuitive enough.

Data Storage in Directory Server

Directory Server manages at least one binary tree database to hold directory data. By default, database files are stored under *instance-path*`/db/`. In general, do not change or move these files.

If you examine the content of the *instance-path*`/db/` directory, you find database log files. You also find subdirectories for each database managed by the server. For instance, *instance-path*`/db/example/` holds data for the directory entries under `dc=example,dc=com`. When you examine the files, you find a number of database indexes, such as `example_sn.db3` for surname attribute values. You also find a `example_id2entry.db3` file containing directory entry information. You can configure Directory Server to encrypt the information in these files if necessary.

From the point of view of client applications, Directory Server presents the directory data stored as directory entries arranged in the directory information tree. Directory Server uses the attribute value indexes to retrieve entries quickly. You can configure which indexes Directory Server maintains.

For instructions on backing up directory data, see Chapter 9, "Directory Server Backup and Restore," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*. For instructions on configuring indexes, see Chapter 13, "Directory Server Indexing," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*. You can also back up directory data and configure indexes using Directory Service Control Center.

Data Replication Between Server Instances

Directory Server allows you to replicate directory data among as many server instances as necessary. Directory Server replication works as an LDAP extended operation that replays update operations from one server to another. The protocol for Directory Server replication is optimized to work quickly over the network. The protocol is also optimized to resolve conflicts when the same data is modified simultaneously on two different server instances.

The unit of Directory Server replication is the suffix. A replication agreement between two servers handles all the directory entries under a base entry in the directory information tree, such as `dc=example,dc=com`. Each agreement to replicate is set up point to point. On one hand, point to point agreements prevent replication from single points of failure when the network becomes partitioned. On the other hand, point to point agreements can be complex to manage as the number of replicas increases. Luckily, Directory Service Control Center handles much of the complexity for you. Directory Service Control Center allows you to manage groups of replicas that provide a common directory service.

You can configure timing, priority, and which data is replicated. You can also configure some servers, called *masters*, to accepts both updates and lookups. You can configure other servers, called *consumers*, to accept only lookups. In addition, you can publish update information over LDAP for client applications that must follow updates as they happen. For further explanation of replication, see Chapter 4, “Directory Server Replication.” For instructions on configuring replication, see Chapter 11, “Directory Server Replication,” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Access Control in Directory Server

Directory Server offers an access control mechanism that works through `aci` attributes placed on directories entries. ACI stands for Access Control Instruction.

ACIs are evaluated based on a user's bind identity. ACIs can be evaluated therefore for all users who can bind to the directory. ACIs can also be applied for anonymous users who did not provide bind credentials. Rules about the bind identity can specify not only which users, but also which systems the users connect from, what time of day they connect, or what authentication method they use.

You configure an ACI to apply to the entries in its scope. Entries that can be in scope include entries on the branch of the directory information tree starting with the entry holding the ACI. Directory Server allows you to configure ACIs to be applied according to a number of different criteria. Directory Server also lets you configure ACIs not only to allow access, but also to deny access.

ACIs can specify which operations are allowed and denied. For example, you typically allow many users to read information, but only a few to update and add directory data.

For further explanation of access control in Directory Server, see "How Directory Server Provides Access Control" on page 42. For instructions on configuring access control, see Chapter 7, "Directory Server Access Control," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

CHAPTER 2

Directory Server Security

For information about the security in Directory Server, see the following sections:

- "How Directory Server Provides Security" on page 41
- "How Directory Server Provides Access Control" on page 42
- "How Directory Server Provides Authentication" on page 67
- "How Directory Server Provides Encryption" on page 88

How Directory Server Provides Security

Directory Server provides security through a combination of the following methods:

- Authentication

  Authentication is a means for one party to verify another's identity. For example, a client gives a password to Directory Server during an LDAP bind operation. Policies define the criteria that a password must satisfy to be considered valid, for example, age, length, and syntax. Directory Server supports anonymous authentication, password-based authentication, certificate-based authentication, SASL-based authentication, and proxy authentication. When authentication is denied, Directory Server provides the following mechanisms to protect data: account inactivation and global lockout. For information about authentication, see "How Directory Server Provides Authentication" on page 67.

- Encryption

  Encryption protects the privacy of information. When data is encrypted, the data is scrambled in a way that only a legitimate recipient can decode. Directory Server supports SSL encryption and attribute encryption. For information about encryption, see "How Directory Server Provides Encryption" on page 88.

- Access control

  Access control tailors the access rights granted to different directory users, and provides a means of specifying required credentials or bind attributes. For information about access control , see "How Directory Server Provides Access Control" on page 42.

- Auditing

  Auditing determines whether the security of a directory has been compromised. For example, log files maintained by a directory can be audited. For information about log files, see Chapter 7, “Directory Server Logging.”

How Directory Server Provides Access Control

Directory Server uses access control instructions (ACIs) to define what rights to grant or deny to requests from LDAP clients. When a directory server receives a request, it uses the ACIs defined in the server, and any authentication information provided by the user to allow or deny access to directory information. The server can allow or deny permissions such as read, write, search, or compare.

For information about ACIs in Directory Server, see the following sections:

- “Introduction to ACIs” on page 42
- “ACI Syntax” on page 45
- “ACI Targets” on page 46
- “ACI Permissions” on page 51
- “ACI Bind Rules” on page 54
- “Tuning and Access Control” on page 66

Introduction to ACIs

ACIs are stored in the `aci` operational attribute. The `aci` attribute is available for use on every entry in the directory, regardless of whether the `aci` attribute is defined for the object class of the entry. The `aci` attribute is multi-valued, therefore multiple ACIs can be defined for the same portion of a directory.

ACIs can be used to control access to the following portions of a directory:

- The entire directory
- A subtree of the directory
- Specific entries in the directory, including entries that define configuration tasks
- A specific set of entry attributes
- Specific entry attribute values

ACIs can be used to define access for the following users:

- A specific user
- All users belonging to a specific group or role
- All users of the directory
- A specific client identified by its IP address or DNS name

Scope and Hierarchy in ACIs

ACIs can be created at any node in a directory tree, including the root DSE.

The scope of an ACI can be the target entry, the target entry and its immediate children, or the target entry and all of its children. When no scope is specified, the ACI applies to the target entry and all of its children.

When a server evaluates access permissions to an entry, it verifies the ACIs for the entry and the ACIs for the parent entries back up to the base of the entry's root suffix. ACIs are not evaluated across chained suffixes on other servers.

Access to an entry in a server must be explicitly granted by an ACI. By default, ACIs define anonymous read access and allow users to modify their own entries, except for attributes needed for security. If no ACI applies to an entry, access is denied to all users except the Directory Manager.

Access granted by an ACI is allowed unless any other ACI in the hierarchy denies it. ACIs that deny access, no matter where they appear in the hierarchy, take precedence over ACIs that allow access to the same resource.

The Directory Manager is the only privileged user to whom access control does not apply. When a client is bound to the directory as the Directory Manager, the server does not evaluate any ACIs before performing operations.

In previous versions of Directory Server, ACIs could not be added or deleted directly under the root DSE. Now this limitation has been removed in Directory Server.

ACI Limitations

The following restrictions apply to ACIs

- If your directory tree is distributed over several servers by using the chaining feature, the following restrictions apply to the use of keywords in access control statements:
  - ACIs that depend on the `groupdn` keyword must be located on the same server as the group entry. If the group is dynamic, then all members of the group must have an entry on the server too. If the group is static, the members' entries can be located on remote servers.
  - ACIs that depend on the `roledn` keyword must be located on the same server as the role definition entry. Every entry that is intended to have the role must also be located on the same server.
  - Attributes generated by a CoS cannot be used in all ACI keywords. Specifically, you should not use attributes generated by CoS with the `userattr` and `userdnattr` keywords because the access control rule will not work.
- Access control rules are always evaluated on the local server. You *must not* specify the hostname or port number of the server in LDAP URLs used in ACI keywords.

- You cannot grant a user the right to proxy as the Directory Manager, nor can you grant proxy rights to the Directory Manager.
- The cache settings used for ensuring that the server fits the physical memory available *do not* apply to ACI caches, which means that an excessive number of ACIs may saturate available memory.

Default ACIs

The following default ACIs are defined on the root DSE:

- All users have anonymous access to the directory for search, compare, and read operations (except for the `userpassword` attribute).
- Bound users can modify their own password.
- Users in the group `cn=Administrators,cn=config` have full access to all entries. This is equivalent to Directory Manager access, although unlike Directory Manager, users in the Administration Group are subject to ACIs.

ACIs and Replication

ACIs are stored as attributes of entries. Therefore, if an entry that contains ACIs is part of a replicated suffix, the ACIs are replicated like any other attribute.

ACIs are always evaluated locally, on the directory server that services the incoming LDAP requests.

When a consumer server receives an update request, the consumer server returns a referral to the master server for evaluation of whether the request can be serviced on the master.

Effective Rights

The effective rights feature can be used to obtain the following information:

- Rights information, including entry level rights, attribute level rights and logging.
- Permissions for write, self write add, and self write delete.
- Logging information for debugging access control problems.

To use the effective rights feature, you must have the access control rights to use the effective rights control and read access to the `aclRights` attribute.

If a proxy control is attached to an effective rights control-based search operation, the effective rights operation is authorized as the proxy user. Therefore the proxy user needs to have the right to use the effective rights control. The entries that the proxy user has the right to search and view are returned.

ACI Syntax

The aci attribute has the following syntax:

```
aci: (target)(version 3.0; acl "name";permission  bindRules;)
```

The following values are used in the ACI syntax:

target — Specifies the entry, attributes, or set of entries and attributes for which you want to control access. The *target* can be a distinguished name, one or more attributes, or a single LDAP filter. The *target* is optional. When a target is not specified, the ACI applies to the entry on which it is defined and its subtree. For information about targets, see “ACI Targets” on page 46.

version 3.0 — A required string that identifies the ACI version.

name — A required string that identifies the ACI. Although there are no restrictions on the name, it is good practice to use unique, descriptive names for ACIs. Using unique names, will allow you to use Get Effective Rights to determine which ACI is in force.

permission — States what rights you are allowing or denying. For information about permissions, see “ACI Permissions” on page 51.

bindRules — Specifies the credentials and bind parameters that a user has to provide to be granted access. Bind rules can also be based on user membership, group membership, or connection properties of the client. For information about bind rules, see “ACI Bind Rules” on page 54.

The permission and bind rule portions of the ACI are set as a pair, also called an Access Control Rule (ACR). The specified permission to access the target is granted or denied depending on whether the accompanying bind rule is evaluated to be true or false.

Multiple targets and multiple permission-bind rule pairs can be used. This allows you to refine both the entry and attributes being targeted and efficiently set multiple access controls for a given target. The following example shows an ACI with multiple targets and multiple permission-bind rule pairs:

```
aci: (targetdefinition)...(targetdefinition)(version 3.0;acl "name";
permission bindRule; ...; permission bindRule;)
```

In the following example, the ACI states that bjensen has rights to modify all attributes in her own directory entry:

```
aci: (target="ldap:///uid=bjensen,dc=example,dc=com"
  (targetattr="*")(targetScope="subtree")(version 3.0; acl "example";
   allow (write) userdn="ldap:///self";)
```

The following sections describe the syntax of targets, permissions and bind rules.

ACI Targets

An ACI target statement specifies the entry, attributes, or set of entries and attributes for which you want to control access.

- “Target Syntax” on page 46
- “Target Keywords” on page 46

Target Syntax

An ACI target statement has this syntax:

(*keyword* = "*expression*")

The following values are used in the target.

keyword — Indicates the type of target.

expression — Identifies the target. The quotation marks ("") around *expression* are syntactically required, although the current implementation accepts expressions like `targetattr=*`.

`=`
`!=` — Indicates that the target is or is not the object specified in the expression.

The `!=` operator cannot be used with the `targettrfilters` keyword or the `targetscope` keyword.

Target Keywords

For a description of target keywords, see the following sections:

- “`target` Keyword” on page 47
- “`targetattr` Keyword” on page 48
- “`targetfilter` Keyword” on page 49
- “`targattrfilters` Keyword” on page 49
- “`targetScope` Keyword” on page 50

The following table lists the target keywords and their associated expressions.

TABLE 2–1 Target Keywords and Their Expressions

Keyword	Type of target	Expression
`target`	A directory entry or its subtree	ldap:///*distinguished_name*
`targetattr`	The attributes of an entry	*attribute*
`targetfilter`	A set of entries or attributes that match an LDAP filter	*LDAP_filter*
`targattrfilters`	An attribute value or combination of values that match an LDAP filter	*LDAP_operation*:*LDAP_filter*
`targetScope`	The scope of the target	`base`, `onelevel`, `subtree`

`target` Keyword

The `target` keyword specifies that an ACI is defined for a directory entry. The `target` keyword uses the following syntax:

```
(target = "distinguished_name")
```

or

```
(target != "distinguished_name")
```

The distinguished name must be in the subtree rooted at the entry where the ACI is defined. For example, the following target may be used in an ACI on `ou=People,dc=example,dc=com`:

```
(target = "ldap:///uid=bjensen,ou=People,dc=example,dc=com")
```

The DN of the entry must be a distinguished name in string representation (RFC 4514). Therefore, characters that are syntactically significant for a DN, such as commas, must be escaped with a single backslash (\).

Wild cards, show as asterisk characters can be used in the expression for the `target` keyword. The asterisk matches an attribute value, a substring of a value, or a DN component. For example, all of the following expressions match `uid=bjensen,ou=people,dc=example,dc=com`.

- `target= "ldap:///uid=bj*,ou=people,dc=example,dc=com"`
- `target= "ldap:///uid=*,ou=people,dc=example,dc=com"`
- `target= "ldap:///*,ou=people,dc=example,dc=com"`
- `target= "ldap:///uid=bjensen,*,dc=com"`
- `target= "ldap:///uid=bjensen*"`

The following further examples show permitted uses of wild cards.

- `target="ldap:///uid=*,dc=example,dc=com"`

  This target matches every entry in the entire `example.com` tree that has the UID attribute in the entry's RDN.
- `target="ldap:///*Anderson,ou=People,dc=example,dc=com"`

  This target matches every entry in the `ou=People` branch whose RDN ends with Anderson, regardless of the naming attribute.
- `target="ldap:///uid=*,ou=*,dc=example,dc=com"`

  This target matches every entry in the `example.com` tree whose distinguished name contains the `uid` and `ou` attributes.

Other usage of wild cards to such as `target="ldap:///uid=bjensen,o*,dc=com"` might be accepted, but are deprecated.

`targetattr` Keyword

The `targetattr` keyword specifies that an ACI is defined for one or more attributes in the targeted entries. The `targetattr` keyword uses the following syntax:

```
(targetattr = "attribute")
```

or

```
(targetattr != "attribute")
```

If no `targetattr` keyword is present, no attributes are targeted. To target all attributes, the `targetattr` keyword must be `targetattr="*"`.

Targeted attributes do not need to exist on the target entry or its subtree, but the ACI applies whenever they do.

Targeted attributes do not need to be defined in the schema. The absence of schema checking makes it possible to implement an access control policy before importing data and its schema.

The `targetattr` keyword can be used for multiple attributes, by using this syntax:

```
(targetattr = "attribute1 || attribute2|| ... attributeN")
```

Note – If you configure attribute aliases, you must specify both the attribute name *and* the alias in the `targetattr` keyword for the ACI to take them into account.

Targeted attributes include all subtypes of the named attribute. For example, `(targetattr = "locality")` also targets `locality;lang-fr`.

Wild cards can be used in the expression for the `targetattr` keyword, but the use of wild cards would serve no purpose and may reduce performance.

`targetfilter` Keyword

The `targetfilter` keyword is used in ACIs to target entries that match an LDAP filter. The ACI applies to all entries that match the LDAP filter *and* that are in the scope of the ACI. The `targetfilter` keyword uses the following syntax:

```
(targetfilter = "(standard LDAP search filter)")
```

EXAMPLE 2–1 Using the `targetfilter` Keyword to Target Specific Entries

The following example is for employees with a status of salaried or contractor, and an attribute for the number of hours worked as a percentage of a full-time position. The filter targets entries for contractors or part-time employees:

```
(targetfilter = "(|(status=contractor)(fulltime<=79))")
```

The Netscape extended filter syntax is not supported in ACIs. For example, the following target filter is not valid:

```
(targetfilter = "(locality:fr:=<= Québec)")
```

The filter syntax that describes matching rules for internationalized values is supported. For example, the following target filter is valid:

```
(targetfilter = "(locality:2.16.840.1.113730.3.3.2.18.1.4:=Québec)")
```

The `targetfilter` keyword selects whole entries as targets of the ACI. The `targetfilter` keyword and the `targetattr` keyword can be used together to create ACIs that apply to a subset of attributes in the targeted entries.

`targattrfilters` Keyword

The `targattrfilters` keyword is used in ACIs to target specific attribute values by using LDAP filters. By using the `targattrfilters` keyword, you can grant or deny permissions on an attribute if that attribute's value meets the criteria defined in the ACI. An ACI that grants or denies access based on an attribute's value, is called a *value-based ACI*. The `targattrfilters` keyword uses this syntax:

```
(targattrfilters="add=attr1:F1 && attr2:F2... && attrn:Fn, \
 del=attr1:F1 && attr2:F2 ... && attrn:Fn")
```

where

`add` represents the operation of creating an attribute.

`del`	represents the operation of deleting an attribute.
`attrn`	represents the target attributes.
`Fn`	represents filters that apply only to the associated attribute.

The following conditions must be met when filters apply to entries, and those entries are created, deleted or modified:

- When an entry is created or deleted, each instance of that attribute must satisfy the filter.
- When an entry is modified, if the operation adds an attribute, then the add filter that applies to that attribute must be satisfied; if the operation deletes an attribute, then the delete filter that applies to that attribute must be satisfied.
- If individual values of an attribute already present in the entry are replaced, then the add and delete filters must be satisfied.

EXAMPLE 2–2 Using the `targattrfilters` Keyword to Allow Users to Add Roles to Their Own Entries

The following ACI allows users to add any role to their own entry, except the `superAdmin` role. It also allows users to add a telephone number with a 123 prefix.

```
(targattrfilters="add=nsroleDN:(!(nsRoleDN=cn=superAdmin)) \
 && telephoneNumber:(telephoneNumber=123*)")
```

`targetScope` Keyword

The `targetScope` keyword is used in ACIs to specify the scope of the ACI. The `targetScope` keyword uses this syntax:

(targetScope="*base*")

The `targetScope` keyword can have one of these values:

`base`	The ACI applies to the target resource only
`onelevel`	The ACI applies to the target resource and its first-generation children
`subtree`	The ACI applies to the target resource and its subtree

If the `targetScope` keyword is not specified, the default value is `subtree`.

ACI Permissions

Permissions specify the type of access that is allowed or denied by the ACI. For information about bind rules, see the following sections:

- "Permission Syntax" on page 51
- "Permission Rights" on page 51
- "Permissions for Typical LDAP Operations" on page 52

Permission Syntax

An ACI permission statement has this syntax:

```
allow|deny (right1, right2 ...)
```

Rights define the operations you can perform on directory data. In an ACI statement, rights is a list of comma-separated keywords enclosed within parentheses.

Rights are granted independently of one another. This means, for example, that a user who is granted `add` rights but not `delete` rights can create an entry but cannot delete an entry. When you are planning the access control policy for your directory, ensure that you grant rights in a way that makes sense for users. For example, it might not make sense to grant `write` permission without granting `read` and `search` permissions.

Permission Rights

The following rights can be allowed or denied in an ACI permission statement:

Read
Permission to read directory data. This permission applies only to the search operation.

Write
Permission to modify an entry by adding, modifying, or deleting attributes. This permission applies to the modify and modify DN operations.

Add
Permission to create entries. This permission applies only to the add operation

Delete
Permission to delete entries. This permission applies only to the delete operation.

Search
Permission to search for directory data. Users must have `Search` and `Read` rights in order to view the data returned as part of a search result. This permission applies only to the search operation.

Compare
Permission for users to compare data they supply with data stored in the directory. With compare rights, the directory returns a success or failure message in response to an inquiry, but the user cannot see the value of the entry or attribute. This permission applies only to the compare operation.

Selfwrite Permission for users to add or delete their own DN in an attribute of the target entry. The syntax of this attribute must be `distinguished name`. This right is used only for group management. The `Selfwrite` permission works with proxy authorization; it grants the right to add or delete the proxy DN from the group entry (not the DN of the bound user).

Proxy Permission for the specified DN to access the target with the rights of another entry. You can grant proxy access using the DN of any user in the directory except the Directory Manager DN. You cannot grant proxy rights to the Directory Manager.

Import Permission for an entry to be imported to the specified DN. This permission applies the modify DN operation.

Export Permission for an entry to be exported from the specified DN. This permission applies the modify DN operation.

All Permission for the specified DN to have the following rights for the targeted entry: `read`, `write`, `search`, `delete`, `compare`, and `selfwrite`. The `All` access right does control permission for the following rights to the target entry: `proxy`, `import`, and `export`.

Permissions for Typical LDAP Operations

This section describes the rights required to perform a set of LDAP operations.

Adding an entry:

- Grant add permission on the entry being added.
- Grant write permission on the value of each attribute in the entry. This right is granted by default but could be restricted using the `targettrfilters` keyword.

Deleting an entry:

- Grant delete permission on the entry to be deleted.
- Grant write permission on the value of each attribute in the entry. This right is granted by default but could be restricted using the `targettrfilters` keyword.

Modifying an attribute in an entry:

- Grant write permission on the attribute type.
- Grant write permission on the value of each attribute type. This right is granted by default but could be restricted using the `targettrfilters` keyword.

Modifying the RDN of an entry:

- Grant write permission on the entry.
- Grant write permission on the attribute type used in the new RDN.

- Grant write permission on the attribute type used in the old RDN, if you want to grant the right to delete the old RDN.
- Grant write permission on the value of attribute type used in the new RDN. This right is granted by default but could be restricted using the `targettrfilters` keyword.

Moving an entry to another subtree:

- Grant export permissions on the entry that you want to move.
- Grant import permission on the new superior entry of the entry that you want to move.

Comparing the value of an attribute:
Grant compare permission on the attribute type.

Searching for entries:

- Grant search permission on each attribute type used in the search filter.
- Grant read permission on at least one attribute type used in the entry to ensure that the entry is returned.
- Grant read permission an each attribute type to be returned with the entry.

EXAMPLE 2–3 Granting ACI Permissions to Perform a Search

This example configures permissions to allow `bjensen` to search her own entry.

```
(target="ldap:///dc=example,dc=com")
ldapsearch -h host -p port -D "uid=bjensen,dc=example,dc=com" \
           -w password -b "dc=example,dc=com" \
             "(objectclass=*)" mail
```

The following ACI determines whether `bjensen` can be granted access for searching her own entry:

```
aci: (targetattr = "mail")(version 3.0; acl "self access to
 mail"; allow (read, search) userdn = "ldap:///self";)
```

The search result list is empty because this ACI does not allow `bjensen` the right to search on the `objectclass` attribute. To perform the search operation described, you must modify the ACI as follows:

```
aci: (targetattr = "mail || objectclass")(version 3.0; acl
 "self access to mail"; allow (read, search) userdn =
 "ldap:///self";)
```

ACI Bind Rules

Bind rules identify the set of users to which an ACI applies. The permission and bind rule portions of the ACI are set as a pair. The specified permission to access the target is granted or denied depending on whether the accompanying bind rule is evaluated to be true or false.

For information about bind rules, see the following sections:

- "Introduction to Bind Rules" on page 54
- "Bind Rule Syntax" on page 54
- "Bind Rule Keywords" on page 55
- "Boolean Bind Rules" on page 66

Introduction to Bind Rules

Bind rules identify a set of users by using the following methods:

- The users, groups, and roles that are granted access.
- The location from which an entity must bind. The location from which a user authenticates can be spoofed and cannot be trusted. Do not base ACIs on this information alone.
- The time or day on which binding must occur.
- The type of authentication that must be in use during binding.

A simple bind rule might require a person accessing the directory to belong to a specific group. A complex bind rule can require a person to belong to a specific group and to log in from a machine with a specific IP address, between 8 am and 5 pm. Additionally, bind rules can be complex constructions that combine these criteria by using Boolean operators.

The server evaluates the logical expressions used in ACIs according to a three-valued logic, similar to the one used to evaluate LDAP filters, as described in section 4.5.1.7 of RFC 4511 *Lightweight Directory Access Protocol (v3)*. Therefore, if any component in the expression evaluates to Undefined (for example if the evaluation of the expression aborted due to a resource limitation), then the server handles this case correctly. The server does not erroneously grant access because an Undefined value occurred in a complex Boolean expression.

Bind Rule Syntax

An ACI bind rule has this syntax:

```
keyword = "expression";
```

or

```
keyword != "expression";
```

The following values are used in the bind rule:

keyword	Indicates the type of bind rule.
expression	Identifies the bind rule.

Bind Rule Keywords

For information about bind rule keywords, see the following sections:

- "userdn Keyword" on page 56
- "Syntax of the userdn Keyword" on page 56
- "LDAP URLs in the userdn Keyword" on page 57
- "groupdn Keyword" on page 58
- "roledn Keyword" on page 58
- "userattr Keyword" on page 59
- "Examples of userattr Keyword With Various Bind Types" on page 59
- "Use of the userattr Keyword With the parent Keyword for Inheritance" on page 61
- "Use of the userattr Keyword to Grant Add Permissions" on page 62
- "ip Keyword" on page 63
- "dns Keyword" on page 64
- "timeofday Keyword" on page 64
- "dayofweek Keyword" on page 65
- "authmethod Keyword" on page 65

The following table summarizes the keywords for bind rules.

TABLE 2–2 Bind Rule Keywords and Their Expressions

Keyword	Used to define access based on	Expression

`userdn`	Specified user	`ldap:///`*distinguished_name*
		`ldap:///all`
		`ldap:///anyone`
		`ldap:///self`
		`ldap:///parent`
		`ldap:///`*suffix*`??sub?(`*filter*`)`

`groupdn`	Specified group or groups	`[ldap:///`*DN*`]`
`roledn`	Specified role or roles	`[ldap:///`*DN*`]`
`userattr`	Matched attribute value	*attribute#bindType* or *attribute#value*
`ip`	Specified IP address or IP addresses	*IP_address*
`dns`	Specified domain or domains	*DNS_host_name*

TABLE 2–2 Bind Rule Keywords and Their Expressions (Continued)

Keyword	Used to define access based on	Expression
`timeofday`	Specified time of day	`0 - 2359`
`dayofweek`	Specified day or days of the week	`sun`, `mon`, `tue`, `wed`, `thu`, `fri`, and `sat`
`authmethod`	Specified authentication method	`none`, `simple`, `ssl`, `sasl` *authentication_method*

`userdn` Keyword

The `userdn` keyword is used to allow or deny access to a specified user. The following sections contain more information about the `userdn` keyword.

Syntax of the `userdn` Keyword

The `userdn` keyword uses this syntax:

```
userdn = "ldap:///dn [|| ldap:///dn]..."
userdn != "ldap:///dn [|| ldap:///dn]..."
```

The `userdn` keyword can alternatively be expressed as an LDAP URL filter. For information about expressing the `userdn` keyword as an LDAP URL, see “LDAP URLs in the `userdn` Keyword” on page 57.

dn can have of the following values:

distinguished-name

A fully qualified DN. Characters that are syntactically significant for a DN, such as commas, must be escaped with a single backslash (`\`). The wild card `*` can be used to specify a set of users. For example, if the following user DN is specified, users with a bind DN beginning with the letter `b` are allowed or denied access:

```
uid=b*,dc=example,dc=com
```

`anyone`

Allows or denies access for anonymous and authenticated users, regardless of the circumstances of the bind.

This access can be limited to specific types of access (for example, access for read or access for search) or to specific subtrees or individual entries within the directory. The following ACI on the `dc=example,dc=com` node allows anonymous access to read and search the entire `dc=example,dc=com` tree.

```
aci: (version 3.0; acl "anonymous-read-search";
 allow (read, search) userdn = "ldap:///anyone";)
```

`all`

Allows or denies access for authenticated users. This `all` value prevents anonymous access. The following ACI on the `dc=example,dc=com` node allows authenticated users to read the entire `dc=example,dc=com` tree:

```
aci: (version 3.0; acl "all-read";
 allow (read) userdn="ldap:///all";)
```

`self`

Allows or denies users access to their own entries if the bind DN matches the DN of the targeted entry. The following ACI on the `dc=example,dc=com` node allows authenticated users in the `dc=example,dc=com` tree to write to their `userPassword` attribute.

```
aci: (targetattr = "userPassword")
 (version 3.0; acl "modify own password"; allow (write)
 userdn = "ldap:///self";)
```

`parent`

Allows or denies users access to the entry if the bind DN is the parent of the targeted entry.

The following ACI on the `dc=example,dc=com` node allows authenticated users in the `dc=example,dc=com` tree to modify any child entries of their bind DN:

```
aci: (version 3.0; acl "parent access";
 allow (write) userdn="ldap:///parent";)
```

LDAP URLs in the `userdn` Keyword

The `userdn` keyword can also be expressed as an LDAP URL with a filter, by using this syntax:

`userdn = ldap:///`*suffix*`??sub?(`*filter*`)`

LDAP URLs always apply to the local server. Do not specify a hostname or port number within an LDAP URL.

The following ACI on the `dc=example,dc=com` node allows all users in the `accounting` and `engineering` branches of the `example.com` tree to access to the targeted resource dynamically based on the following URL

```
userdn = "ldap:///dc=example,dc=com??sub?(|(ou=eng)(ou=acct))"
```

LDAP URLs can be used with the logical OR operator and the not-equal operator as shown in the following examples.

EXAMPLE 2–4 Using the `userdn` Keyword With a Logical OR Operator in LDAP URLs

This bind rule is evaluated to be true for users that bind with either of the specified DN patterns.

```
userdn = "ldap:///uid=b*,c=example.com || ldap:///cn=b*,dc=example,dc=com";
```

EXAMPLE 2–5 Using the `userdn` Keyword With a Not-Equal Operator in LDAP URLs

This bind rule is evaluated to be true if the client is not binding as a UID-based DN in the accounting subtree. This bind rule only makes sense if the targeted entry is not under the accounting branch of the directory tree.

```
userdn != "ldap:///uid=*,ou=Accounting,dc=example,dc=com";
```

`groupdn` Keyword

The `groupdn` keyword specifies that access to a targeted entry is granted or denied if the user binds by using a DN that belongs to a specific group. The `groupdn` keyword uses this syntax:

```
groupdn="ldap:///groupDN [|| ldap:///groupDN]..."
```

The bind rule is evaluated to be true if the bind DN belongs to a group that is specified by any of the values for *groupDN*.

In the following example, the bind rule is true if the bind DN belongs to the Administrators group :

```
aci: (version 3.0; acl "Administrators-write"; allow (write)
 groupdn="ldap:///cn=Administrators,dc=example,dc=com";)
```

Characters that are syntactically significant for a DN, such as commas, must be escaped with a single backslash (\).

`roledn` Keyword

The `roledn` keyword specifies that access to a targeted entry is granted or denied if the user binds using a DN that belongs to a specific role. The `roledn` keyword requires one or more valid distinguished names, in this format:

```
roledn = "ldap:///dn [|| ldap:///dn]... [|| ldap:///dn]"
```

The bind rule is evaluated to be true if the bind DN belongs to the specified role.

Characters that are syntactically significant for a DN, such as commas, must be escaped with a single backslash (\).

The `roledn` keyword has the same syntax and is used in the same way as the `groupdn` keyword.

`userattr` Keyword

The `userattr` keyword specifies which attribute values in the entry that was used to bind must match those in the targeted entry. The `userattr` keyword can be used for the following attributes:

- User DN
- Group DN
- Role DN
- LDAP filter, in an LDAP URL
- Any attribute type

An attribute generated by a Class of Service (CoS) definition cannot be used with the `userattr` keyword. ACIs that contain bind rules that depend on attribute values generated by CoS will not work.

The `userattr` keyword uses this syntax:

```
userattr = "attrName#bindType"
```

Alternatively, if you are using an attribute type that requires a value other than a user DN, group DN, role DN, or an LDAP filter, the `userattr` keyword uses this syntax:

```
userattr = "attrName#attrValue"
```

The `userattr` keyword can have one of the following values:

attrName	The name of the attribute used for value matching
bindType	One of the following types of bind: `USERDN,GROUPDN,ROLEDN,LDAPURL`
attrValue	Any string that represents an attribute value

Examples of `userattr` Keyword With Various Bind Types

EXAMPLE 2–6 Using the `userattr` Keyword With the `USERDN` Bind Type

The following is an example of the `userattr` keyword associated with a bind based on the user DN:

```
userattr = "manager#USERDN"
```

EXAMPLE 2–6 Using the `userattr` Keyword With the `USERDN` Bind Type *(Continued)*

The bind rule is evaluated to be true if the bind DN matches the value of the `manager` attribute in the targeted entry. You can use this to allow a user's manager to modify employees' attributes. This mechanism only works if the `manager` attribute in the targeted entry is expressed as a full DN.

The following example grants a manager full access to his or her employees' entries:

```
aci: (target="ldap:///dc=example,dc=com")(targetattr="*")
 (version 3.0;acl "manager-write";
 allow (all) userattr = "manager#USERDN";)
```

EXAMPLE 2–7 Using the `userattr` Keyword With the `GROUPDN` Bind Type

The following is an example of the `userattr` keyword associated with a bind based on a group DN:

```
userattr = "owner#GROUPDN"
```

The bind rule is evaluated to be true if the bind DN is a member of the group specified in the `owner` attribute of the targeted entry. For example, you can use this mechanism to allow a group to manage employees' status information. You can use an attribute other than `owner`, as long as the attribute you use contains the DN of a group entry.

The group you point to can be a dynamic group, and the DN of the group can be under any suffix in the directory. However, the evaluation of this type of ACI by the server is very resource intensive.

If you are using static groups that are under the same suffix as the targeted entry, you can use the following expression:

```
userattr = "ldap:///dc=example,dc=com?owner#GROUPDN"
```

In this example, the group entry is under the `dc=example,dc=com` suffix. The server can process this type of syntax more quickly than the previous example.

EXAMPLE 2–8 Using the `userattr` Keyword With the `ROLEDN` Bind Type

The following is an example of the `userattr` keyword associated with a bind based on a role DN:

```
userattr = "exampleEmployeeReportsTo#ROLEDN"
```

EXAMPLE 2–8 Using the `userattr` Keyword With the `ROLEDN` Bind Type *(Continued)*

The bind rule is evaluated to be true if the bind DN belongs to the role specified in the `exampleEmployeeReportsTo` attribute of the targeted entry. For example, if you create a nested role for all managers in your company, you can use this mechanism to grant managers at all levels access to information about employees that are at a lower grade than themselves.

The DN of the role can be under any suffix in the directory. If, in addition, you are using filtered roles, the evaluation of this type of ACI uses a lot of resources on the server.

EXAMPLE 2–9 Using the `userattr` Keyword With the `LDAPURL` Bind Type

The following is an example of the `userattr` keyword associated with a bind based on an LDAP filter:

```
userattr = "myfilter#LDAPURL"
```

The bind rule is evaluated to be true if the bind DN matches the filter specified in the *myfilter* attribute of the targeted entry. The *myfilter* attribute can be replaced by any attribute that contains an LDAP filter.

EXAMPLE 2–10 Using the `userattr` Keyword With Any Attribute Value

The following is an example of the `userattr` keyword associated with a bind based on any attribute value:

```
userattr = "favoriteDrink#Milk"
```

The bind rule is evaluated to be true if the bind DN and the target DN include the `favoriteDrink` attribute with a value of **`Milk`**.

Use of the `userattr` Keyword With the `parent` Keyword for Inheritance

The `userattr` keyword can be used with the `parent` keyword to specify the number of levels below the target that should inherit the ACI. The `userattr` keyword and `parent` keyword use this syntax:

```
userattr = "parent[inheritance_level].attribute#bindType"
```

The `userattr` keyword and `parent` keyword can have the following values:

inheritance_level A comma separated list that indicates how many levels below the target should inherit the ACI. These levels below the targeted entry can be specified: `[0,1,2,3,4]`. Zero (0) indicates the targeted entry.

attribute	The attribute targeted by the `userattr` or `groupattr` keyword.
bindType	The type of bind can be `USERDN` or `GROUPDN`. Inheritance cannot be used with `LDAPURL` and `ROLEDN` binds.

The following example shows how the `userattr` keyword is used with the `parent` keyword for inheritance:

```
userattr = "parent[0,1].manager#USERDN"
```

This bind rule is evaluated to be true if the `bindDN` matches the manager attribute of the targeted entry. The permissions granted when the bind rule is evaluated to be true apply to the target entry *and* to all entries immediately below it.

Use of the `userattr` Keyword to Grant Add Permissions

If you use the `userattr` keyword in conjunction with `all` or `add` permissions, you might find that the behavior of the server is not what you expect. Typically, when a new entry is created in the directory, Directory Server evaluates access rights on the entry being created, and not on the parent entry. However, for ACIs that use the `userattr` keyword, this behavior could create a security hole, and the server's normal behavior is modified to avoid it.

Consider the following example:

```
aci: (target="ldap:///dc=example,dc=com")(targetattr="*")
 (version 3.0; acl "manager-write"; allow (all)
 userattr = "manager#USERDN";)
```

This ACI grants managers all rights on the entries of employees that report to them. However, because access rights are evaluated on the entry being created, this type of ACI would also allow any employee to create an entry in which the manager attribute is set to their own DN. For example, disgruntled employee Joe (`cn=Joe,ou=eng,dc=example,dc=com`), might want to create an entry in the Human Resources branch of the tree, to use (or misuse) the privileges granted to Human Resources employees.

He could do this by creating the following entry:

```
dn: cn= Trojan Horse,ou=Human Resources,dc=example,dc=com
objectclass: top
...
cn: Trojan Horse
manager: cn=Joe,ou=eng,dc=example,dc=com
```

To avoid this type of security threat, the ACI evaluation process does not grant add permission at level 0, that is, to the entry itself. You can, however, use the `parent` keyword to grant add rights below existing entries. You must specify the number of levels below the parent for add rights. For example, the following ACI allows child entries to be added to any entry in the dc=example,dc=com that has a manager attribute that matches the bind DN:

```
aci: (target="ldap:///dc=example,dc=com")(targetattr="*")
 (version 3.0; acl "parent-access"; allow (add)
 userattr = "parent[0,1].manager#USERDN";)
```

This ACI ensures that add permission is granted only to users whose bind DN matches the manager attribute of the parent entry.

ip Keyword

The ip keyword is used to specify that a bind operation must originate from a specific IP address. The ip keyword uses this syntax:

```
ip = "IPaddressList" or ip != "IPaddressList"
```

The *IPaddressList* value is a list of one or more comma-separated elements from the following elements:

- A specific IPv4 address: 123.45.6.7
- An IPv4 address with wild cards to specify a subnetwork: 12.3.45.*
- An IPv4 address or subnetwork with subnetwork mask: 123.45.6.*, 255.255.255.0
- An IPv6 address in any of its legal forms and contained in square brackets [and], as defined by RFC 2373 (http://www.ietf.org/rfc/rfc2373.txt).

  The following addresses are equivalent:

  - 12AB:0000:0000:CD30:0000:0000:0000:0000
  - 12AB::CD30:0:0:0:0
  - 12AB:0:0:CD30::
- An IPv6 address with a subnet prefix length: 12AB::CD30:0:0:0:0/60

The bind rule is evaluated to be true if the client accessing the directory is located at the named IP address.

The ip keyword can be used to force all directory updates to occur from a given machine or network domain. However, the IP address from which a user authenticates can be spoofed, and can therefore not be trusted. Do not base ACIs on this information alone.

The wild card * can be used to specify a set of IP addresses.

The wild card * cannot be used in IPv6 addresses.

`dns` Keyword

Note – The `dns` keyword requires that the naming service used on your machine is DNS. If the name service is not DNS, you should use the `ip` keyword instead.

The `dns` keyword is used to specify that a bind operation must originate from a specific domain or host machine. The `dns` keyword uses this syntax:

`dns = "`*DNS_Hostname*`" or dns != "`*DNS_Hostname*`"`

The `dns` keyword requires a fully qualified DNS domain name. Granting access to a host without specifying the domain creates a potential security threat. For example, the following expression is allowed but not recommended:

```
dns = "legend.eng";
```

You should use a fully qualified name such as:

```
dns = "legend.eng.example.com";
```

The `dns` keyword allows wild cards.

```
dns = "*.example.com";
```

The bind rule is evaluated to be true if the client accessing the directory is located in the named domain. This can be useful for allowing access only from a specific domain. Note that wild cards do not work if your system uses a naming service other than DNS.

`timeofday` Keyword

The `timeofday` keyword is used to specify that access can occur at a certain time of day. The time and date on the server are used for the evaluation of the `timeofday` and `dayofweek` bind rules, and not the time on the client. The `timeofday` keyword uses this syntax:

`timeofday` *operator* "*time*"

The `timeofday` keyword can have the following values:

operator

- Equal to (=)
- Not equal to (`!=`)
- Greater than or equal to (>=)
- Less than (<)
- Less than or equal to (<=)

time Four digits representing hours and minutes in the 24-hour clock (`0` to `2359`)

- `timeofday = "1200";` is true if the client is accessing the directory during the minute that the system clock shows noon.
- `timeofday != "0100";` is true for access at any other time than 1 a.m.
- `timeofday > "0800";` is true for access from 8:01 a.m. through 11:59 p.m.
- `timeofday >= "0800";` is true for access from 8:00 a.m. through 11:59 p.m.
- `timeofday < "1800";` is true for access from 12:00 midnight through 5:59 p.m.

`dayofweek` Keyword

The `dayofweek` keyword is used to specify that access can occur on a certain day or on certain days of the week. The time and date on the server are used for the evaluation of the `timeofday` and `dayofweek` bind rules, and not the time on the client. The `dayofweek` keyword uses this syntax:

`dayofweek = "`*day1, day2 ...*`"`

The bind rule is true if the directory is being accessed on one of the days listed.

The `dayofweek` keyword can have one or more of the following values: `sun`, `mon`, `tue`, `wed`, `thu`, `fri`, `sat`.

`authmethod` Keyword

The `authmethod` keyword is used to specify that a client must bind to the directory by using a specific authentication method. The `authmethod` keyword uses this syntax:

`authmethod = "`*authentication_method*`"`

The `authmethod` keyword can have the following values:

None
Authentication is not checked during bind rule evaluation. This is the default.

Simple
The bind rule is evaluated to be true if the client is accessing the directory using a username and password.

SSL
The client must bind to the directory over a Secure Sockets Layer (SSL) or Transport Layer Security (TLS) connection.

The bind rule is evaluated to be true if the client authenticates to the directory by using a certificate over LDAPS. It will not be true if the client authenticates by using simple authentication (bind DN and password) over LDAPS.

SASL *sasl_mechanism* The bind rule is evaluated to be true if the client authenticates to the directory by using one of the following SASL mechanisms: `DIGEST-MD5`, `GSSAPI`, or `EXTERNAL`.

Boolean Bind Rules

Bind rules can be complex expressions that use the Boolean expressions `AND`, `OR`, and `NOT` to set precise access rules. Boolean bind rules use this syntax:

```
(bindRuleA and (bindRuleB or (bindRuleC and bindRuleD));)
```

Parentheses defines the order in which rules are evaluated, and a trailing semicolon must appear after the final rule.

EXAMPLE 2–11 Boolean Bind Rule

The bind rule is true if both of the following conditions are met:

- The bind DN client is accessed from within the `example.com` domain
- The bind DN client is a member of either the administrators group *or* the bind DN client a member of both the mail administrators and calendar administrators groups

```
(dns = "*.example.com"
  and (groupdn = "ldap:///cn=administrators, dc=example,dc=com"
  or (groupdn = "ldap:///cn=mail administrators, dc=example,dc=com"
  and groupdn = "ldap:///cn=calendar administrators, dc=example,dc=com"));)
```

Tuning and Access Control

Directory Server offers performance and scalability improvements for Access Control Instructions. The improvements include better memory management. The improvements also include support for macro ACIs. Improvements notwithstanding, Directory Server uses significant system resources to evaluate complex ACIs. Extensive use of complex ACIs can therefore negatively impact performance.

Macro ACIs help you limit the number of ACIs used. By limiting the number of ACIs, you render access control easier to manage and reduce the load on the system. Macros are placeholders that represent a DN, or a portion of a DN, in an ACI. A macro can be used in an ACI target, in an ACI bind rule, or in both. When Directory Server receives a request, it checks which ACI macros match against the resource targeted for the resulting operation. If a macro matches, Directory Server replaces it with the value of the actual DN. Directory Server then evaluates the ACI normally.

Testing has demonstrated that a Directory Server instance can support more than 50,000 ACIs. Nevertheless, keep the number of ACIs as small as possible. Keeping the number of ACIs small limits negative impact on performance. Keeping the number small also reduces the complexity of managing access controls. For deployments involving complex ACI environments, consider using Directory Proxy Server to provide some access control features.

How Directory Server Provides Authentication

Authentication is the process of confirming an identity. In network interactions, authentication involves the confident identification of one party by another party. Network interactions typically take place between a client, such as browser software running on a personal computer, and a server, such as the software and hardware used to host a Web site. *Client authentication* refers to the confident identification of a client by a server; *server authentication* refers to the confident identification of a server by a client.

For information about authentication, see the following sections:

- "Anonymous Access" on page 67
- "Password-Based Authentication" on page 68
- "Certificate-based Authentication" on page 70
- "SASL-based Authentication" on page 85
- "Proxy Authorization" on page 86
- "Account Inactivation" on page 87
- "Global Account Lockout" on page 87

Anonymous Access

Anonymous access lets a user bind to the directory without providing authentication credentials. With access control, you can give anonymous users whatever privileges you choose. Often, anonymous users are allowed to read non-sensitive data from the directory, such as names, telephone numbers, and email addresses.

You can also restrict the privileges of anonymous access, or limit anonymous access to a subset of attributes that contain address book information. Anonymous access should not be allowed for sensitive data.

In cases where anonymous users have access to something, you may want to prevent users who fail to bind properly nevertheless being granted access as anonymous. See the `require-bind-pwd-enabled` in `server`(5dsconf) for more information.

Password-Based Authentication

Simple password authentication offers an easy way of authenticating users. In password authentication, the user must supply a password for each server, and the administrator must keep track of the name and password for each user, typically on separate servers.

Steps in Password-Based Authentication

Figure 2–1 shows the steps involved in authenticating a client by using a name and password. The figure assumes the following points.

- The user has already decided to trust the system, either without authentication, or on the basis of server authentication via SSL.
- The user has requested a resource controlled by the server.
- The server requires client authentication before permitting access to the requested resource.

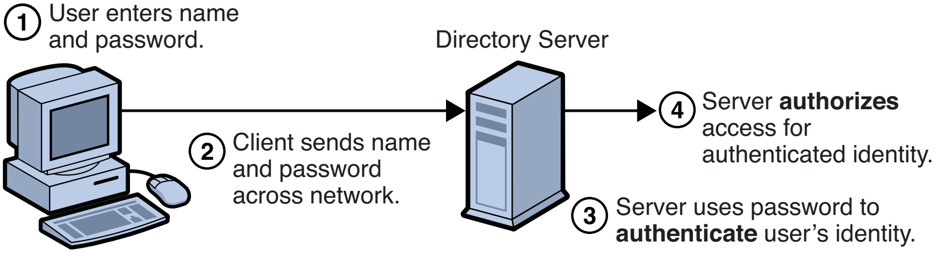

1 User enters name and password.
Directory Server
2 Client sends name and password across network.
4 Server **authorizes** access for authenticated identity.
3 Server uses password to **authenticate** user's identity.

FIGURE 2–1 Password-Based Authentication

In Figure 2–1, password authentication is performed in the following steps.

1. The user enters a name and password.

   For the LDAP bind to Directory Server, the client application must bind with a Distinguished Name. Therefore the client application may use the name entered by the user to retrieve the DN.
2. The client sends the DN and password across the network.
3. The server determines whether the password sent from the client matches the password stored for the entry with the DN sent from the client.

   If so, the server accepts the credentials as evidence authenticating the user identity.
4. The server determines whether the identified user is permitted to access the requested resource.

   If so, the server allows the client to access the resource.

Password Policy

A password policy is a set of rules that govern how passwords are administered in a system. Directory Server supports multiple password policies. The password policy can be configured to suit the security requirements of your deployment.

Instances of Directory Server are created with a default password policy.

Types of Password Policy

Directory Server provides the following password policies.

Default password policy

The default password policy is defined in the configuration entry `cn=PasswordPolicy,cn=config`. The default password policy applies to all accounts in the directory except for the directory manager.

The parameters of the default policy can be modified to override the default settings. However, because the default password policy is part of the configuration for the instance, modifications to the default password policy cannot be replicated.

Specialized password policy

A password policy can be configured for an individual user or for set of users by using the CoS and roles features. However, specialized password policies can not be applied to static groups.

A specialized password policy is defined in a subentry in the directory tree. Like the default password policy, the specialized password policy uses the `pwdPolicy` object class. For example, the following entry defines a specialized password policy:

```
dn: cn=TempPolicy,dc=example,dc=com
objectClass: top
objectClass: pwdPolicy
objectClass: LDAPsubentry
cn: TempPolicy
pwdCheckQuality: 2
pwdLockout: on
pwdLockoutDuration: 300
pwdMaxFailure: 3
pwdMustChange: on
```

A specialized password policy can be assigned to a single user account or can be assigned to a set of users by using roles. For example, in the following entry the password policy defined in `cn=TempPolicy,dc=example,dc=com` is assigned to the `pwdPolicySubentry` attribute of the user entry:

```
dn: uid=dmiller,ou=people,dc=example,dc=com
objectClasaccess controls: person
```

```
objectClass: top
sn: miller
cn: david
userPassword: secret12
pwdPolicySubentry: cn=TempPolicy,dc=example,dc=com
```

When referenced by a user entry, a specialized password policy overrides the default password policy.

Because specialized password policies are defined the directory data, they can be replicated.

Configuration of Password Policy

For information about how to configure password policy, see Chapter 8, "Directory Server Password Policy," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

For information about the attributes used to configure password policies, see the `pwpolicy`(5dssd) man page.

Certificate-based Authentication

For information about client authentication with certificates, see the following sections:

- "Introduction to Certificate-based Authentication" on page 70
- "Steps for Configuring Certificate-based Authentication" on page 72
- "Certificates and Certificate Authorities (CA)" on page 72
- "Types of Certificates" on page 79
- "Contents of a Certificate" on page 80
- "Certificate Management" on page 83

Introduction to Certificate-based Authentication

Figure 2–2 shows how certificates and the SSL protocol are used together for authentication. To authenticate a user to a server, a client digitally signs a randomly generated piece of data and sends both the certificate and the signed data across the network. For the purposes of this discussion, the digital signature associated with some data can be thought of as evidence provided by the client to the server. The server authenticates the user's identity on the strength of this evidence.

Like for password-based authentication illustrated in Figure 2–1, Figure 2–2 assumes that the user has already decided to trust the server and has requested a resource. The server has requested client authentication in the process of evaluating whether to grant access to the requested resource.

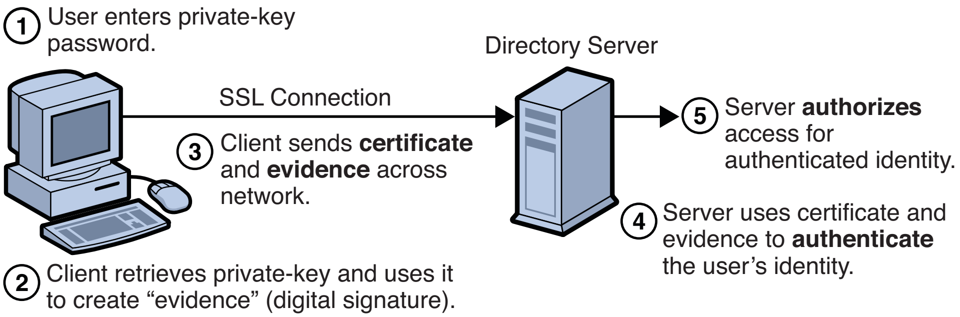

1 User enters private-key password.
Directory Server
SSL Connection
5 Server **authorizes** access for authenticated identity.
3 Client sends **certificate** and **evidence** across network.
4 Server uses certificate and evidence to **authenticate** the user's identity.
2 Client retrieves private-key and uses it to create "evidence" (digital signature).

FIGURE 2–2 Certificate-Based Authentication

Unlike for password-based authentication illustrated in Figure 2–1, Figure 2–2 requires the use of SSL. In Figure 2–2 it is assumed that the client has a valid certificate that can be used to identify the client to the server.

Certificate-based authentication is generally considered preferable to password-based authentication because it is based on what the user has, the private key, as well as what the user knows, the password that protects the private key. However, it's important to note that these two assumptions are true only if unauthorized personnel have not gained access to the user's machine or password, the password for the client software's private key database has been set, and the software is set up to request the password at reasonably frequent intervals.

Note – Neither password-based authentication nor certificate-based authentication address security issues related to physical access to individual machines or passwords. Public-key cryptography can only verify that a private key used to sign some data corresponds to the public key in a certificate. It is the user's responsibility to protect a machine's physical security and to keep the private-key password secret.

Certificates replace the authentication portion of the interaction between the client and the server. Instead of requiring a user to send passwords across the network throughout the day, single sign-on requires the user to enter the private-key database password just once, without sending it across the network. For the rest of the session, the client presents the user's certificate to authenticate the user to each new server it encounters. Existing authorization mechanisms based on the authenticated user identity are not affected.

Steps for Configuring Certificate-based Authentication

In Figure 2–2, certificate-based authentication is set in the following steps.

1. The client software maintains a database of the private keys that correspond to the public keys published in any certificates issued for that client. The client asks for the password to this database the first time the client needs to access it during a given session—for example, the first time the user attempts to access an SSL-enabled server that requires certificate-based client authentication. After entering this password once, the user doesn't need to enter it again for the rest of the session, even when accessing other SSL-enabled servers.
2. The client unlocks the private-key database, retrieves the private key for the user's certificate, and uses that private key to digitally sign some data that has been randomly generated for this purpose on the basis of input from both the client and the server. This data and the digital signature constitute "evidence" of the private key's validity. The digital signature can be created only with that private key and can be validated with the corresponding public key against the signed data, which is unique to the SSL session.
3. The client sends both the user's certificate and the evidence, the randomly generated piece of data that has been digitally signed, across the network.
4. The server uses the certificate and the evidence to authenticate the user's identity.
5. At this point the server may optionally perform other authentication tasks, such as checking that the certificate presented by the client is stored in the user's entry in an LDAP directory. The server then continues to evaluate whether the identified user is permitted to access the requested resource. This evaluation process can employ a variety of standard authorization mechanisms, potentially using additional information in an LDAP directory, company databases, and so on. If the result of the evaluation is positive, the server allows the client to access the requested resource.

Certificates and Certificate Authorities (CA)

A certificate is an electronic document that identifies an individual, a server, a company, or some other entity. A certificate also associates that identity with a public key. Like a driver's license, a passport, or other commonly used personal IDs, a certificate provides generally recognized proof of someone's or something's identity.

Certificate authorities, CAs, validate identities and issue certificates. CAs can be independent third parties or organizations that run their own certificate-issuing server software. The methods used to validate an identity vary depending on the policies of a given CA. In general, before issuing a certificate, the CA must use its published verification procedures for that type of certificate to ensure that an entity requesting a certificate is in fact who it claims to be.

A certificate issued by a CA binds a particular public key to the name of the entity the certificate identifies, such as the name of an employee or a server. Certificates help prevent the use of fake public keys for impersonation. Only the public key certified by the certificate works with the corresponding private key possessed by the entity identified by the certificate.

In addition to a public key, a certificate always includes the name of the entity it identifies, an expiration date, the name of the CA that issued the certificate, a serial number, and other information. Most importantly, a certificate always includes the digital signature of the issuing CA. The CA's digital signature allows the certificate to function as a "letter of introduction" for users who know and trust the CA but don't know the entity identified by the certificate.

Any client or server software that supports certificates maintains a collection of trusted CA certificates. These CA certificates determine which other certificates the software can validate, in other words, which issuers of certificates the software can trust. In the simplest case, the software can validate only certificates issued by one of the CAs for which it has a certificate. It's also possible for a trusted CA certificate to be part of a chain of CA certificates, each issued by the CA above it in a certificate hierarchy.

For information about CAs, see the following sections:

- "CA Hierarchies" on page 73
- "Certificate Chains" on page 74
- "Verifying a Certificate Chain" on page 76

CA Hierarchies

In large organizations, it may be appropriate to delegate the responsibility for issuing certificates to several different certificate authorities. For example, the number of certificates required may be too large for a single CA to maintain; different organizational units may have different policy requirements; or it may be important for a CA to be physically located in the same geographic area as the people to whom it is issuing certificates.

It's possible to delegate certificate-issuing responsibilities to subordinate CAs. The X.509 standard includes a model for setting up a hierarchy of CAs.

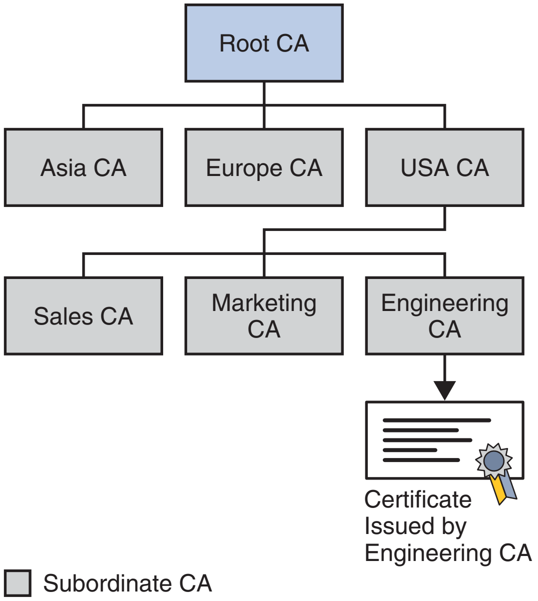


FIGURE 2–3 Hierarchy of Certificate Authorities

In this model, the root CA is at the top of the hierarchy. The root CA's certificate is a *self-signed certificate*. That is, the certificate is digitally signed by the same entity, the root CA, that the certificate identifies. The CAs that are directly subordinate to the root CA have CA certificates signed by the root CA. CAs under the subordinate CAs in the hierarchy have their CA certificates signed by the higher-level subordinate CAs.

Organizations have a great deal of flexibility in terms of the way they set up their CA hierarchies. Figure 2–3 shows just one example; many other arrangements are possible.

Certificate Chains

CA hierarchies are reflected in certificate chains. A *certificate chain* is a series of certificates issued by successive CAs. Figure 2–4 shows a certificate chain leading from a certificate that identifies some entity through two subordinate CA certificates to the CA certificate for the root CA (based on the CA hierarchy shown in the following figure).

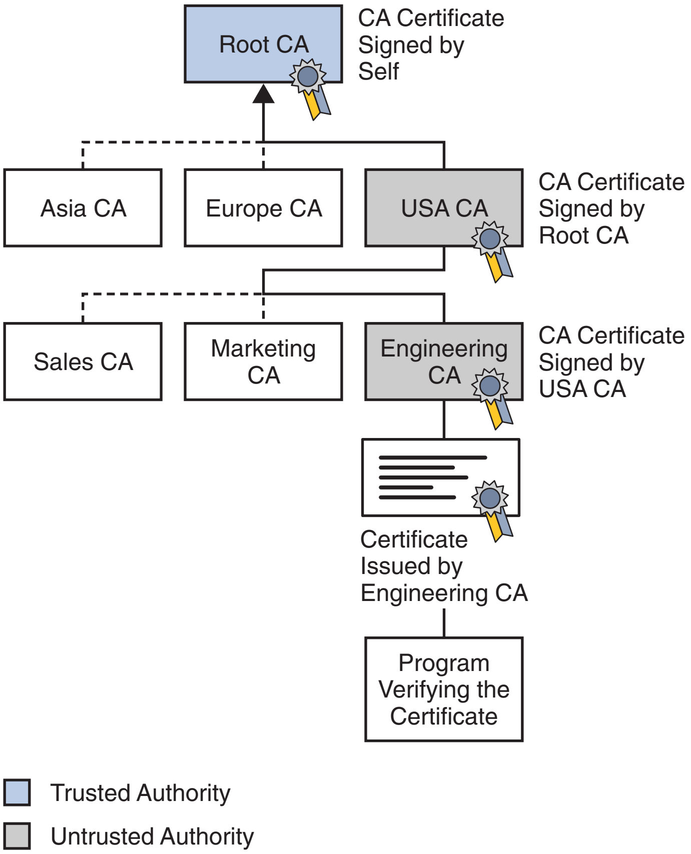

Root CA
CA Certificate Signed by Self
Asia CA
Europe CA
USA CA
CA Certificate Signed by Root CA
Sales CA
Marketing CA
Engineering CA
CA Certificate Signed by USA CA
Certificate Issued by Engineering CA
Program Verifying the Certificate
Trusted Authority
Untrusted Authority

FIGURE 2–4 Certificate Chain

A certificate chain traces a path of certificates from a branch in the hierarchy to the root of the hierarchy. In a certificate chain, the following occur:

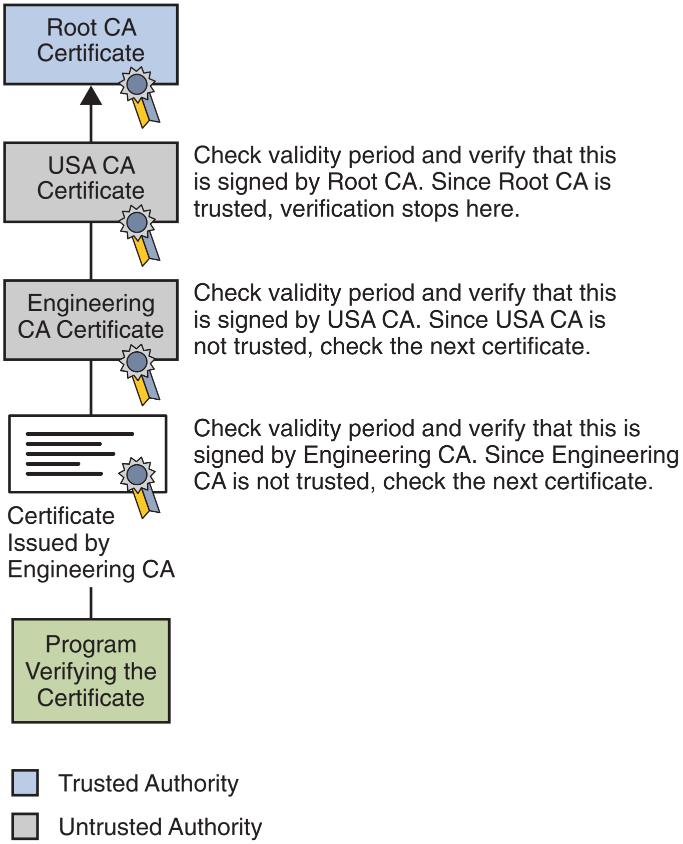
- Each certificate is followed by the certificate of its issuer.
- In Figure 2–4, the Engineering CA certificate contains the DN of the CA (that is, USA CA), that issued that certificate. USA CA's DN is also the subject name of the next certificate in the chain.
- Each certificate is signed with the private key of its issuer. The signature can be verified with the public key in the issuer's certificate, which is the next certificate in the chain.

In Figure 2–4, the public key in the certificate for the USA CA can be used to verify the USA CA's digital signature on the certificate for the Engineering CA.

Verifying a Certificate Chain

Certificate chain verification is the process of making sure a given certificate chain is well-formed, valid, properly signed, and trustworthy. Directory Server software uses the following steps to form and verify a certificate chain, starting with the certificate being presented for authentication:

1. The certificate validity period is checked against the current time provided by the verifier's system clock.
2. The issuer's certificate is located. The source can be either the verifier's local certificate database (on that client or server) or the certificate chain provided by the subject (for example, over an SSL connection).
3. The certificate signature is verified using the public key in the issuer certificate.
4. If the issuer's certificate is trusted by the verifier in the verifier's certificate database, verification stops successfully here. Otherwise, the issuer's certificate is checked to make sure it contains the appropriate subordinate CA indication in the Directory Server certificate type extension, and chain verification returns to step 1 to start again, but with this new certificate.


Root CA Certificate
USA CA Certificate
Engineering CA Certificate
Certificate Issued by Engineering CA
Program Verifying the Certificate
Trusted Authority
Untrusted Authority

Check validity period and verify that this is signed by Root CA. Since Root CA is trusted, verification stops here.

Check validity period and verify that this is signed by USA CA. Since USA CA is not trusted, check the next certificate.

Check validity period and verify that this is signed by Engineering CA. Since Engineering CA is not trusted, check the next certificate.

FIGURE 2–5 Verifying A Certificate Chain

Figure 2–5 shows what happens when only Root CA is included in the verifier's local database. If a certificate for one of the intermediate CAs shown in Figure 2–6, such as Engineering CA, is found in the verifier's local database, verification stops with that certificate, as shown in the following figure.

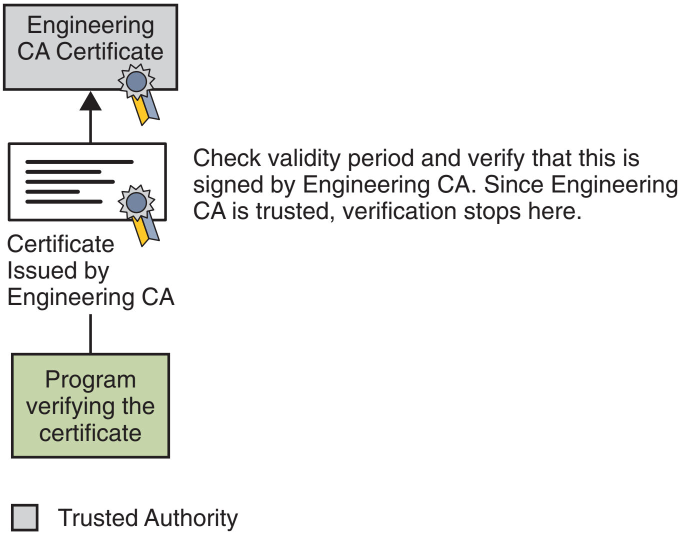

Engineering CA Certificate
Certificate Issued by Engineering CA
Check validity period and verify that this is signed by Engineering CA. Since Engineering CA is trusted, verification stops here.
Program verifying the certificate
Trusted Authority

FIGURE 2–6 Verifying A Certificate Chain to an Intermediate CA

Expired validity dates, an invalid signature, or the absence of a certificate for the issuing CA at any point in the certificate chain causes authentication to fail. For example, the following figure shows how verification fails if neither the Root CA certificate nor any of the intermediate CA certificates are included in the verifier's local database.

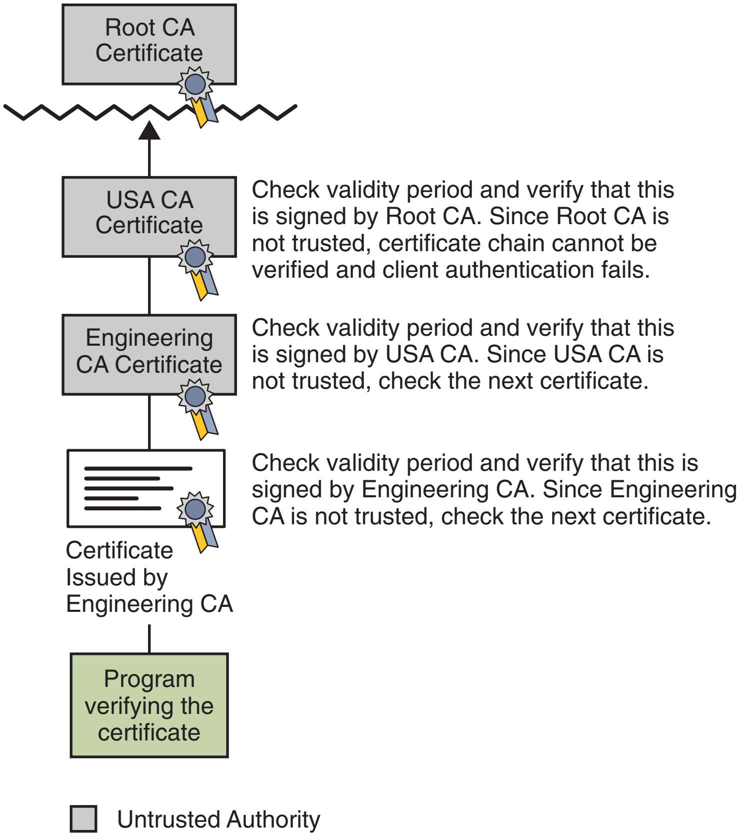

Root CA Certificate
USA CA Certificate
Check validity period and verify that this is signed by Root CA. Since Root CA is not trusted, certificate chain cannot be verified and client authentication fails.
Engineering CA Certificate
Check validity period and verify that this is signed by USA CA. Since USA CA is not trusted, check the next certificate.
Check validity period and verify that this is signed by Engineering CA. Since Engineering CA is not trusted, check the next certificate.
Certificate Issued by Engineering CA
Program verifying the certificate
Untrusted Authority

FIGURE 2–7 Certificate Chain That Cannot Be Verified

For general information about the way digital signatures work, see "Digital Signatures" on page 99.

Types of Certificates

Directory Server uses the following types of certificate:

Client SSL certificates

Client SSL certificates are used to identify clients to servers via SSL (client authentication). Typically, the identity of the client is assumed to be the same as the identity of a human being, such as an employee in an enterprise. Client SSL certificates can also be used for form signing and as part of a single sign-on solution.

For example, a bank gives a customer a client SSL certificate that allows the bank's servers to identify that customer and authorize access to the customer's accounts. A company might give a new employee a client SSL certificate that allows the company's servers to identify that employee and authorize access to the company's servers.

Server SSL certificates

Server SSL certificates are used to identify servers to clients via SSL (server authentication). Server authentication may be used with or without client authentication. Server authentication is a requirement for an encrypted SSL session.

For example, internet sites that engage in electronic commerce usually support certificate-based server authentication, at a minimum, to establish an encrypted SSL session and to assure customers that they are dealing with a web site identified with a particular company. The encrypted SSL session ensures that personal information sent over the network, such as credit card numbers, cannot easily be intercepted.

S/MIME certificates

S/MIME certificates are used for signed and encrypted email. As with client SSL certificates, the identity of the client is typically assumed to be the same as the identity of a human being, such as an employee in an enterprise. A single certificate may be used as both an S/MIME certificate and an SSL certificate. S/MIME certificates can also be used for form signing and as part of a single sign-on solution.

For example, a company deploys combined S/MIME and SSL certificates solely for the purpose of authenticating employee identities, thus permitting signed email and client SSL authentication but not encrypted email. Another company issues S/MIME certificates solely for the purpose of both signing and encrypting email that deals with sensitive financial or legal matters.

Object-signing certificates

Object-signing certificates are used to identify signers of Java code, JavaScript scripts, or other signed files.

For example, a software company signs software distributed over the Internet to provide users with some assurance that the software is a legitimate product of that company. Using certificates and digital signatures in this manner can also make it possible for users to identify and control the kind of access downloaded software has to their computers.

CA certificates

CA certificates are used to identify CAs. Client and server software use CA certificates to determine what other certificates can be trusted.

For example, the CA certificates stored in client software determine what other certificates that client can authenticate. An administrator can implement some aspects of corporate security policies by controlling the CA certificates stored in each user's client.

Contents of a Certificate

The contents of certificates supported by Directory Server and many other software companies are organized according to the X.509 v3 certificate specification, which has been recommended by the International Telecommunications Union (ITU), an international standards body, since 1988. Examples in this section show samples of the data and signature sections of a certificate.

Every X.509 certificate consists of the following sections.

- A data section, including the following information.
  - The version number of the X.509 standard supported by the certificate.
  - The certificate's serial number. Every certificate issued by a CA has a serial number that is unique among the certificates issued by that CA.
  - Information about the user's public key, including the algorithm used and a representation of the key itself.
  - The DN of the CA that issued the certificate.
  - The period during which the certificate is valid (for example, between 1:00 p.m. on November 15, 2003 and 1:00 p.m. November 15, 2004).
  - The DN of the certificate subject (for example, in a client SSL certificate this would be the user's DN), also called the subject name.
  - Optional *certificate extensions,* which may provide additional data used by the client or server. For example, the certificate type extension indicates the type of certificate—that is, whether it is a client SSL certificate, a server SSL certificate, a certificate for signing email, and so on. Certificate extensions can also be used for a variety of other purposes.
- A signature section, includes the following information.
  - The cryptographic algorithm, or cipher, used by the issuing CA to create its own digital signature.
  - The CA's digital signature, obtained by hashing all of the data in the certificate together and encrypting it with the CA's private key.

EXAMPLE 2–12 Data and Signature Sections of a Certificate in Human-Readable Format

```
Certificate:
Data:
   Version: v3 (0x2)
   Serial Number: 3 (0x3)
   Signature Algorithm: PKCS #1 MD5 With RSA Encryption
   Issuer: OU=Certificate Authority, O=Example Industry, C=US
   Validity:
    Not Before: Fri Oct 17 18:36:25 2003
    Not  After: Sun Oct 17 18:36:25 2004
   Subject: CN=Jane Doe, OU=Finance, O=Example Industry, C=US
   Subject Public Key Info:
    Algorithm: PKCS #1 RSA Encryption
    Public Key:
       Modulus:
          00:ca:fa:79:98:8f:19:f8:d7:de:e4:49:80:48:e6:2a:2a:86:
```

EXAMPLE 2–12 Data and Signature Sections of a Certificate in Human-Readable Format *(Continued)*

```
         ed:27:40:4d:86:b3:05:c0:01:bb:50:15:c9:de:dc:85:19:22:
         43:7d:45:6d:71:4e:17:3d:f0:36:4b:5b:7f:a8:51:a3:a1:00:
         98:ce:7f:47:50:2c:93:36:7c:01:6e:cb:89:06:41:72:b5:e9:
         73:49:38:76:ef:b6:8f:ac:49:bb:63:0f:9b:ff:16:2a:e3:0e:
         9d:3b:af:ce:9a:3e:48:65:de:96:61:d5:0a:11:2a:a2:80:b0:
         7d:d8:99:cb:0c:99:34:c9:ab:25:06:a8:31:ad:8c:4b:aa:54:
         91:f4:15
      Public Exponent: 65537 (0x10001)
  Extensions:
   Identifier: Certificate Type
     Critical: no
     Certified Usage:
     SSL Client
   Identifier: Authority Key Identifier
     Critical: no
     Key Identifier:
       f2:f2:06:59:90:18:47:51:f5:89:33:5a:31:7a:e6:5c:fb:36:
       26:c9
  Signature:
   Algorithm: PKCS #1 MD5 With RSA Encryption
  Signature:
6d:23:af:f3:d3:b6:7a:df:90:df:cd:7e:18:6c:01:69:8e:54:65:fc:06:
30:43:34:d1:63:1f:06:7d:c3:40:a8:2a:82:c1:a4:83:2a:fb:2e:8f:fb:
f0:6d:ff:75:a3:78:f7:52:47:46:62:97:1d:d9:c6:11:0a:02:a2:e0:cc:
2a:75:6c:8b:b6:9b:87:00:7d:7c:84:76:79:ba:f8:b4:d2:62:58:c3:c5:
b6:c1:43:ac:63:44:42:fd:af:c8:0f:2f:38:85:6d:d6:59:e8:41:42:a5:
4a:e5:26:38:ff:32:78:a1:38:f1:ed:dc:0d:31:d1:b0:6d:67:e9:46:a8:
d:c4
```

EXAMPLE 2–13 Certificate In the 64-Byte Encoded Form Interpreted by Software

```
-----BEGIN CERTIFICATE-----
MIICKzCCAZSgAwIBAgIBAzANBgkqhkiG9w0BAQQFADA3MQswCQYDVQQGEwJVUzER
MA8GA1UEChMITmV0c2NhcGUxFTATBgNVBAsTDFN1cHJpeWEncyBDQTAeFw05NzEw
MTgwMTM2MjVaFw05OTEwMTgwMTM2MjVaMEgxCzAJBgNVBAYTAlVTMREwDwYDVQQK
EwhOZXRzY2FwZTENMAsGA1UECxMEUHViczEXMBUGA1UEAxMOU3Vwcml5YSBTaGV0
dHkwgZ8wDQYJKoZIhvcNAQEFBQADgY0AMIGJAoGBAMr6eZiPGfjX3uRJgEjmKiqG
7SdATYazBcABu1AVyd7chRkiQ31FbXFOGD3wNktbf6hRo6EAmM5/R1AskzZ8AW7L
iQZBcrXpc0k4du+2Q6xJu2MPm/8WKuMOnTuvzpo+SGXelmHVChEqooCwfdiZywyZ
NMmrJgaoMa2MS6pUkfQVAgMBAAGjNjA0MBEGCWCGSAGG+EIBAQQEAwIAgDAfBgNV
HSMEGDAWgBTy8gZZkBhHUfWJM1oxeuZc+zYmyTANBgkqhkiG9w0BAQQFAAOBgQBt
I6/z07Z635DfzX4XbAFpjlRl/AYwQzTSYx8GfcNAqCqCwaSDKvsuj/vwbf91o3j3
UkdGYpcd2cYRCgKi4MwqdWyLtpuHAH18hHZ5uvi00mJYw8W2wUOsY0RC/a/IDy84
hW3WWehBUqVK5SY4/zJ4oTjx7dwNMdGwbWfpRqjd1A==
-----END CERTIFICATE-----
```

Certificate Management

The set of standards and services that facilitate the use of public-key cryptography and X.509 v3 certificates in a network environment is called the*public key infrastructure* (PKI). For information about the certificate management issues addressed by Directory Server, see the following sections:

- "Issuing Certificates" on page 83
- "Certificates and the LDAP Directory" on page 83
- "Key Management" on page 84
- "Renewal and Revocation of Certificates" on page 84
- "Registration Authorities" on page 85

Issuing Certificates

The process for issuing a certificate depends on the certificate authority that issues it and the purpose for which it is used. The process for issuing non-digital forms of identification varies in similar ways. For example, if you want to get a generic ID card (not a driver's license) from the Department of Motor Vehicles in California, the requirements are straightforward: you need to present some evidence of your identity, such as a utility bill with your address on it and a student identity card. If you want to get a regular driving license, you also need to take a test — a driving test when you first get the license, and a written test when you renew it. If you want to get a commercial license for an eighteen-wheeler, the requirements are much more stringent. If you live in some other state or country, the requirements for various kinds of licenses differ.

Similarly, different CAs have different procedures for issuing different kinds of certificates. In some cases the only requirement may be your mail address. In other cases, your UNIX login and password may be sufficient. At the other end of the scale, for certificates that identify people who can authorize large expenditures or make other sensitive decisions, the issuing process may require notarized documents, a background check, and a personal interview.

Depending on an organization's policies, the process of issuing certificates can range from being completely transparent for the user to requiring significant user participation and complex procedures. In general, processes for issuing certificates should be highly flexible, so organizations can tailor them to their changing needs.

Issuing certificates is one of several management tasks that can be handled by separate Registration Authorities.

Certificates and the LDAP Directory

The Lightweight Directory Access Protocol (LDAP) for accessing directory services supports great flexibility in the management of certificates within an organization. System administrators can store much of the information required to manage certificates in an LDAP-compliant directory. For example, a CA can use information in a directory to pre-populate a certificate with a new employee's legal name and other information. The CA can leverage directory information in other ways to issue certificates one at a time or in bulk, using a range of different identification techniques depending on the security policies of a given organization. Other routine management tasks, such as key management and renewing and revoking certificates, can be partially or fully automated with the aid of the directory.

Information stored in the directory can also be used with certificates to control access to various network resources by different users or groups. Issuing certificates and other certificate management tasks can thus be an integral part of user and group management.

Key Management

Before a certificate can be issued, the public key it contains and the corresponding private key must be generated. Sometimes it may be useful to issue a single person one certificate and key pair for signing operations, and another certificate and key pair for encryption operations. Separate signing and encryption certificates make it possible to keep the private signing key on the local machine only, thus providing maximum nonrepudiation, and to back up the private encryption key in some central location where it can be retrieved in case the user loses the original key or leaves the company.

Keys can be generated by client software or generated centrally by the CA and distributed to users via an LDAP directory. There are trade-offs involved in choosing between local and centralized key generation. For example, local key generation provides maximum nonrepudiation, but may involve more participation by the user in the issuing process. Flexible key management capabilities are essential for most organizations.

Key recovery, or the ability to retrieve backups of encryption keys under carefully defined conditions, can be a crucial part of certificate management (depending on how an organization uses certificates). Key recovery schemes usually involve an *m of n* mechanism: for example, *m* of *n* managers within an organization might have to agree, and each contribute a special code or key of their own, before a particular person's encryption key can be recovered. This kind of mechanism ensures that several authorized personnel must agree before an encryption key can be recovered.

Renewal and Revocation of Certificates

Like a driver's license, a certificate specifies a period of time during which it is valid. Attempts to use a certificate for authentication before or after its validity period fails. Therefore, mechanisms for managing certificate renewal are essential for any certificate management strategy. For example, an administrator may wish to be notified automatically when a certificate is about to expire, so that an appropriate renewal process can be completed in plenty of time without causing the certificate's subject any inconvenience. The renewal process may involve reusing the same public-private key pair or issuing a new one.

A driver's license can be suspended even if it has not expired—for example, as punishment for a serious driving offense. Similarly, it's sometimes necessary to revoke a certificate before it has expired—for example, if an employee leaves a company or moves to a new job within the company.

Certificate revocation can be handled in several different ways. For some organizations, it may be sufficient to set up servers so that the authentication process includes checking the directory for the presence of the certificate being presented. When an administrator revokes a certificate, the certificate can be automatically removed from the directory, and subsequent authentication attempts with that certificate fails even though the certificate remains valid in every other respect. Another approach involves publishing a certificate revocation list (CRL)—that is, a list of revoked certificates—to the directory at regular intervals and checking the list as part of the authentication process. For some organizations, it may be preferable to check directly with the issuing CA each time a certificate is presented for authentication. This procedure is sometimes called real-time status checking.

Registration Authorities

Interactions between entities identified by certificates (sometimes called end entities) and CAs are an essential part of certificate management. These interactions include operations such as registration for certification, certificate retrieval, certificate renewal, certificate revocation, and key backup and recovery. In general, a CA must be able to authenticate the identities of end entities before responding to the requests. In addition, some requests need to be approved by authorized administrators or managers before being serviced.

As previously discussed, the means used by different CAs to verify an identity before issuing a certificate can vary widely, depending on the organization and the purpose for which the certificate is used. To provide maximum operational flexibility, interactions with end entities can be separated from the other functions of a CA and handled by a separate service called a *Registration Authority* RA.

An RA acts as a front end to a CA by receiving end entity requests, authenticating them, and forwarding them to the CA. After receiving a response from the CA, the RA notifies the end entity of the results. RAs can be helpful in scaling a PKI across different departments, geographical areas, or other operational units with varying policies and authentication requirements.

SASL-based Authentication

Client authentication during an SSL or TLS connection can also use the Simple Authentication and Security Layer (SASL). Directory Server supports the following SASL mechanisms.

DIGEST-MD5 The DIGEST-MD5 mechanism authenticates clients by comparing a hashed value sent by the client with a hash of the user's password. However, because the mechanism must read user passwords, all users wishing to be authenticated through DIGEST-MD5 must have clear text passwords in the directory.

GSSAPI GSSAPI is available on the Solaris Operating System only. The General Security Services API (GSSAPI) allows Directory Server to interact with the Kerberos V5 security system to identify a user. The client application must present its credentials to the Kerberos system, which in turn validates the user's identity to Directory Server.

For information about how to configure SASL-based authentication, see “Configuring Credential Levels and Authentication Methods” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Proxy Authorization

Proxy authorization allows requests from clients to be processed with a proxy identity instead of the identity of the client. A client, binding with its own identity is granted, through proxy authorization, the rights of a proxy user. The Access Control Instructions (ACIs) of the proxy user, not the ACIs of the client, are evaluated to allow or deny the operation.

Before performing an operation with proxy authorization, the account of the proxy user is validated. If the proxy user account is locked out, inactivated, if the password has been reset or has expired the client operation is aborted.

By using proxy authorization, an LDAP application can use a single bind to service multiple users who are making requests against Directory Server. Instead of having to bind and authenticate for each user, the client application binds to Directory Server and uses proxy rights.

The following conditions must be satisfied in order to use proxy authorization:

- The Directory Server must be configured with appropriate ACIs for the proxy identity.

  For example, the following ACI gives the administrator the `ALL` access right:

  ```
  aci: (targetattr="*") (version 3.0; acl "allowAll-Admin";
    allow (all) userdn="ldap:///uid=Administrator,
    ou=Administrators, dc=example,dc=com";)
  ```
- The Directory Server must be configured with permission for proxy identity to act as the proxy for other users.

  For example, the following ACI gives the administrator the right to act as the proxy for the user `ClientApplication`:

  ```
  aci: (targetattr="*") (version 3.0; acl "allowproxy-
    accountingsoftware"; allow (proxy) userdn=
    "ldap:///dn:uid=ClientApplication,ou=Applications,
    dc=example,dc=com";)
  ```

The following sample shows the user `ClientApplication` performing a search operation by using the `Administrator` proxy identity:

```
$ ldapsearch \
-D "uid=ClientApplication,ou=Applications,dc=example,dc=com" \
-w password \
-y "uid=Administrator,ou=Administrators,dc=example,dc=com" ...
```

Note that the client binds as itself, but is granted the privileges of the proxy entry. The client does not need the password of the proxy entry.

Proxy rights can be granted to any user except the Directory Manager.

For information about how to configure proxy authorization, see "Proxy Authorization" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Account Inactivation

A user account or a set of accounts can be inactivated temporarily or indefinitely by using the `ns-inactivate`(1M) command. When the account is inactivated, the user cannot bind to Directory Server. This feature is called *account inactivation*.

User accounts and roles can be inactivated. When a role is inactivated, the members of the role are inactivated, not the role itself.

For information about how to configure account inactivation, see "Manually Locking Accounts" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Global Account Lockout

Depending on the password policy settings, a client account can be locked out of an account when the number of failed bind attempts exceeds the number of allowed bind attempts. In a replicated topology the client is locked out of all instances of Directory Server, not just the instance to which the client was attempting to bind. This feature is called *global account lockout*.

In versions of Directory Server prior to Directory Server 6, account lockout was based on integer counters. By default, these counters were not replicated.

In this version of the product, bind failures are recorded by using timestamps. By default, the timestamps are replicated, and prioritized replication is used to replicate updates to the lockout data that are caused by failed bind requests.

Global account lockout can be used in the following scenarios:

- When replication is used to propagate bind failures

  Bind requests must not be directed to read-only consumers. When a client fails to bind to a read-only consumer, the lockout data is not replicated. Therefore, if a bind request fails on a read-only consumer, the lockout data is updated on that instance only and is not replicated across the topology.

  Even if all bind attempts are directed at master replicas, the client might be able to perform bind attempts on multiple servers faster than the lockout data can be replicated. In this way, a client can exceed the limit on failed bind attempts for the password policy. Note that this risk is present even though bind failures are replicated by using prioritized replication.
- When Directory Proxy Server manages the routing of bind operations

  The Directory Proxy Server can achieve global account lockout by using the hash algorithm for load-balancing to route all bind requests for a given account to the same Directory Server. For information about using the hash algorithm for global account lockout, see "Operational Affinity Algorithm for Global Account Lockout" on page 266.

How Directory Server Provides Encryption

For information about how Directory Server encrypts data, see the following sections:

- "Secure Sockets Layer (SSL)" on page 88
- "Digital Signatures" on page 99
- "Key Encryption" on page 101
- "Attribute Encryption" on page 103

Secure Sockets Layer (SSL)

SSL provides encrypted communications and optional authentication between a Directory Server and its clients. SSL can be used over LDAP or DSML over HTTP. SSL is enabled by default over LDAP and can be enabled for DSML over HTTP.

Replication can be configured to use SSL for secure communications between servers. When replication is configured to use SSL, data sent to and from the server is encrypted by using SSL.

By default, Directory Server allows simultaneous unsecured and secure communications, suing different port numbers. Unsecured LDAP communications are handled on one port, conventionally port number 389. Secure LDAP communications are handled on another port, conventionally port number 636.

For security reasons, you can also restrict all communications to the secure port. Client authentication is also configurable. You can set client authentication to required or allowed. This setting determines the level of security you enforce.

SSL enables support for the Start TLS extended operation that provides security on a regular LDAP connection. Clients can bind to the non-SSL port and then use the Transport Layer Security protocol to initiate an SSL connection. The Start TLS operation allows more flexibility for clients, and can help simplify port allocation.

For information about SSL, see the following sections:

- "Overview of SSL" on page 89
- "Cryptographic Algorithms Used With SSL" on page 90
- "SSL Handshake" on page 91

Overview of SSL

TCP/IP governs the transport and routing of data over the Internet. Other protocols, such as the HTTP, LDAP, or IMAP use TCP/IP to support typical application tasks such as displaying web pages or running mail servers.

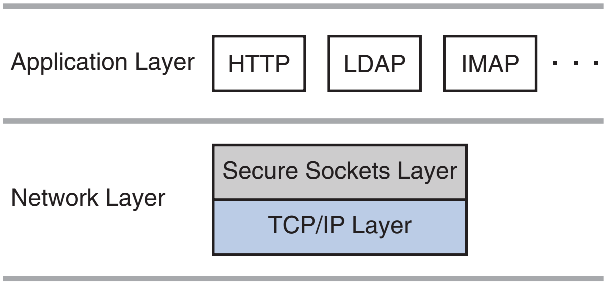


FIGURE 2–8 Where SSL Runs

The SSL protocol runs above TCP/IP and below higher-level protocols such as HTTP or IMAP. It uses TCP/IP on behalf of the higher-level protocols, and in the process allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection.

SSL addresses the following concerns about communication over the Internet and other TCP/IP networks:

SSL server authentication allows a user to confirm a server's identity.

> SSL-enabled client software can use standard techniques of public-key cryptography to check that a server's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the client's list of trusted CAs. This confirmation might be important if the user, for example, is sending a credit card number over the network and wants to check the receiving server's identity.

SSL client authentication allows a server to confirm a user's identity.

Using the same techniques as those used for server authentication, SSL-enabled server software can check that a client's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the server's list of trusted CAs. This confirmation might be important if the server, for example, is a bank sending confidential financial information to a customer and wants to check the recipient's identity.

An encrypted SSL connection requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software, thus providing a high degree of confidentiality.

Confidentiality is important for both parties to any private transaction. In addition, all data sent over an encrypted SSL connection is protected with a mechanism for detecting tampering—that is, for automatically determining whether the data has been altered in transit.

The SSL protocol includes two sub-protocols: the SSL record protocol and the SSL handshake protocol.

The SSL record protocol defines the format used to transmit data. The SSL handshake protocol involves using the SSL record protocol to exchange a series of messages between an SSL-enabled server and an SSL-enabled client when they first establish an SSL connection. This exchange of messages is designed to facilitate the following actions:

- Authenticate the server to the client.
- Allow the client and server to select the cryptographic algorithms, or ciphers, that they both support.
- Optionally authenticate the client to the server.
- Use public-key encryption techniques to generate shared secrets.
- Establish an encrypted SSL connection.

For more information about the handshake process, see "SSL Handshake" on page 91.

Cryptographic Algorithms Used With SSL

Cipher suites define the following aspects of SSL communication:

- The key exchange Algorithm
- The encryption cipher
- The encryption cipher key length
- The message authentication method

The SSL protocol supports many ciphers. Clients and servers can support different cipher suites, depending on factors such as the version of SSL they support, and company policies regarding acceptable encryption strength. The SSL handshake protocol determines how the server and client negotiate which cipher suites they use to authenticate each other, to transmit certificates, and to establish session keys.

SSL 2.0 and SSL 3.0 protocols support overlapping sets of cipher suites. Administrators can enable or disable any of the supported cipher suites for both clients and servers. When a client and server exchange information during the SSL handshake, they identify the strongest enabled cipher suites they have in common and use those for the SSL session. Decisions about which cipher suites to enable depend on the sensitivity of the data involved, the speed of the cipher, and the applicability of export rules.

Key-exchange algorithms like KEA and RSA govern the way in which a server and client determine the symmetric keys they use during an SSL session. The most commonly used SSL cipher suites use the RSA key exchange.

The list of ciphers enabled for Directory Server, and also the list of ciphers supported by Directory Server can be obtained with the `dsconf` command. For information about using the `dsconf` command to list available ciphers and manage ciphers, see "Choosing Encryption Ciphers" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Support for ciphers is provided by the Network Security Services, NSS, component. For details about NSS, see theNSS project site (`http://www.mozilla.org/projects/security/pki/nss/`).

SSL Handshake

The SSL protocol uses a combination of public-key and symmetric key encryption. Symmetric key encryption is much faster than public-key encryption, but public-key encryption provides better authentication techniques. An SSL session always begins with an exchange of messages called the *SSL handshake*. The handshake allows the server to authenticate itself to the client by using public-key techniques, and then allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection. Optionally, the handshake also allows the client to authenticate itself to the server.

For information about the SSL handshake, see the following sections:

- "Messages Exchanged During SSL Handshake" on page 91
- "Server Authentication During SSL Handshake" on page 93
- "Man-In-the-Middle Attack" on page 95
- "Client Authentication During SSL Handshake" on page 96

Messages Exchanged During SSL Handshake

The following steps describes the sequence of messages exchanged during an SSL handshake. These step describe the programmatic details of the messages exchanged during the SSL handshake.

1. The client sends the server the client's SSL version number, cipher settings, randomly generated data, and other information the server needs to communicate with the client using SSL.

2. The server sends the client the server's SSL version number, cipher settings, randomly generated data, and other information the client needs to communicate with the server over SSL. The server also sends its own certificate and, if the client is requesting a server resource that requires client authentication, requests the client's certificate.
3. The client uses some of the information sent by the server to authenticate the server. For details, see "Server Authentication During SSL Handshake" on page 93. If the server cannot be authenticated, the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client goes on to Step 4.
4. Using all data generated in the handshake so far, the client, with the cooperation of the server, depending on the cipher being used, creates the pre-master secret for the session, encrypts it with the server's public key, obtained from the server's certificate, sent in Step 2, and sends the encrypted pre-master secret to the server.
5. If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case the client sends both the signed data and the client's own certificate to the server along with the encrypted pre-master secret.
6. If the server has requested client authentication, the server attempts to authenticate the client. For details, see "Server Authentication During SSL Handshake" on page 93. If the client cannot be authenticated, the session is terminated. If the client can be successfully authenticated, the server uses its private key to decrypt the pre-master secret, then performs a series of steps (which the client also performs, starting from the same pre-master secret) to generate the master secret.
7. Both the client and the server use the master secret to generate the *session keys*, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity—that is, to detect changes in the data between the time it was sent and the time it is received over the SSL connection.
8. The client sends a message to the server informing it that future messages from the client are encrypted with the session key. It then sends a separate (encrypted) message indicating that the client portion of the handshake is finished.
9. The server sends a message to the client informing it that future messages from the server are encrypted with the session key. It then sends a separate (encrypted) message indicating that the server portion of the handshake is finished.
10. The SSL handshake is now complete, and the SSL session has begun. The client and the server use the session keys to encrypt and decrypt the data they send to each other and to validate its integrity.

Before continuing with a session, directory servers can be configured to check that the client's certificate is present in the user's entry in an LDAP directory. This configuration option provides one way of ensuring that the client's certificate has not been revoked.

Both client and server authentication involve encrypting some piece of data with one key of a public-private key pair and decrypting it with the other key:

- In the case of server authentication, the client encrypts the pre-master secret with the server's public key. Only the corresponding private key can correctly decrypt the secret, so the client has some assurance that the identity associated with the public key is in fact the server with which the client is connected. Otherwise, the server cannot decrypt the pre-master secret and cannot generate the symmetric keys required for the session, and the session is terminated.
- In the case of client authentication, the client encrypts some random data with the client's private key—that is, it creates a digital signature. The public key in the client's certificate can correctly validate the digital signature only if the corresponding private key was used. Otherwise, the server cannot validate the digital signature and the session is terminated.

Server Authentication During SSL Handshake

SSL-enabled client software always requires server authentication, or cryptographic validation by a client of the server's identity. The server sends the client a certificate to authenticate itself. The client uses the certificate to authenticate the identity the certificate claims to represent.

To authenticate the binding between a public key and the server identified by the certificate that contains the public key, an SSL-enabled client must receive a `yes` answer to the four questions shown in the following figure.

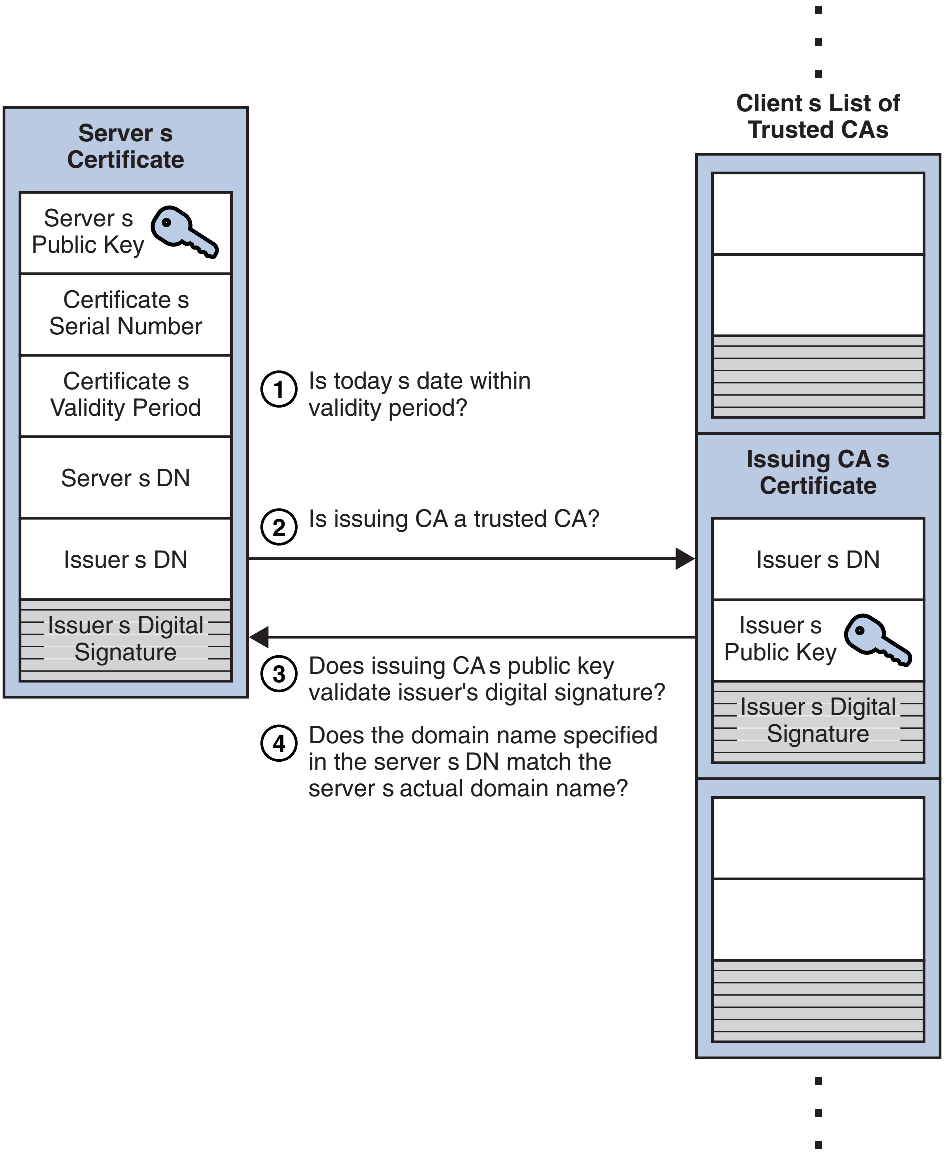

Server s Certificate
Server s Public Key
Certificate s Serial Number
Certificate s Validity Period
Server s DN
Issuer s DN
Issuer s Digital Signature
1 Is today s date within validity period?
2 Is issuing CA a trusted CA?
3 Does issuing CA s public key validate issuer's digital signature?
4 Does the domain name specified in the server s DN match the server s actual domain name?
Client s List of Trusted CAs
Issuing CA s Certificate
Issuer s DN
Issuer s Public Key
Issuer s Digital Signature

FIGURE 2–9 Authenticating a Client Certificate During SSL Handshake

An SSL-enabled client goes through the following steps to authenticate a server's identity:

1. Is today's date within the validity period?

The client checks the server certificate's validity period. If the current date and time are outside of that range, the authentication process won't go any further. If the current date and time are within the certificate's validity period, the client goes on to the next step.

2. Is the issuing CA a trusted CA?

   Each SSL-enabled client maintains a list of trusted CA certificates, represented by the shaded area on the right—hand side of Figure 2–9. This list determines which server certificates the client accepts. If the distinguished name (DN) of the issuing CA matches the DN of a CA on the client's list of trusted CAs, the answer to this question is yes, and the client goes on to the next step. If the issuing CA is not on the list, the server is not authenticated unless the client can verify a certificate chain ending in a CA that is on the list.

3. Does the issuing CA's public key validate the issuer's digital signature?

   The client uses the public key from the CA's certificate (which it found in its list of trusted CAs in step 2) to validate the CA's digital signature on the server certificate being presented. If the information in the server certificate has changed since it was signed by the CA or if the CA certificate's public key doesn't correspond to the private key used by the CA to sign the server certificate, the client won't authenticate the server's identity. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA and proceeds. At this point, the client has determined that the server certificate is valid.

4. Does the domain name in the server's certificate match the domain name of the server itself?

   This step confirms that the server is actually located at the same network address specified by the domain name in the server certificate. Although step 4 is not technically part of the SSL protocol, it provides the only protection against a form of security attack known as *man-in-the-middle*. Clients must perform this step and must refuse to authenticate the server or establish a connection if the domain names don't match. If the server's actual domain name matches the domain name in the server certificate, the client goes on to the next step.

5. The server is authenticated.

   The client proceeds with the SSL handshake. If the client doesn't get to step 5 for any reason, the server identified by the certificate cannot be authenticated, and the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server requires client authentication, the server performs the steps described in "Client Authentication During SSL Handshake" on page 96.

After the steps described here, the server must successfully use its private key to decrypt the pre-master secret sent by the client.

Man-In-the-Middle Attack

The *man-in-the-middle* is a rogue program that intercepts all communication between the client and a server with which the client is attempting to communicate via SSL. The rogue program intercepts the legitimate keys that are passed back and forth during the SSL handshake, substitutes its own, and makes it appear to the client that it is the server, and to the server that it is the client.

The encrypted information exchanged at the beginning of the SSL handshake is actually encrypted with the rogue program's public key or private key, rather than the client's or server's real keys. The rogue program ends up establishing one set of session keys for use with the real server, and a different set of session keys for use with the client. This allows the rogue program not only to read all the data that flows between the client and the real server, but also to change the data without being deleted. Therefore, it is extremely important for the client to check that the domain name in the server certificate corresponds to the domain name of the server with which a client is attempting to communicate—in addition to checking the validity of the certificate by performing the other steps described in "Server Authentication During SSL Handshake" on page 93

Client Authentication During SSL Handshake

SSL-enabled servers can be configured to require client authentication, or cryptographic validation by the server of the client's identity. When a server configured this way requests client authentication separate piece of digitally signed data to authenticate itself. The server uses the digitally signed data to validate the public key in the certificate and to authenticate the identity the certificate claims to represent.

The SSL protocol requires the client to create a digital signature by creating a one-way hash from data generated randomly during the handshake and known only to the client and server. The hash of the data is then encrypted with the private key that corresponds to the public key in the certificate being presented to the server.

To authenticate the binding between the public key and the person or other entity identified by the certificate that contains the public key, an SSL-enabled server must receive a yes answer to the first four questions shown in Figure 2–10. Although the fifth question is not part of the SSL protocol, directory servers can be configured to support this requirement to take advantage of the user entry in an LDAP directory as part of the authentication process.

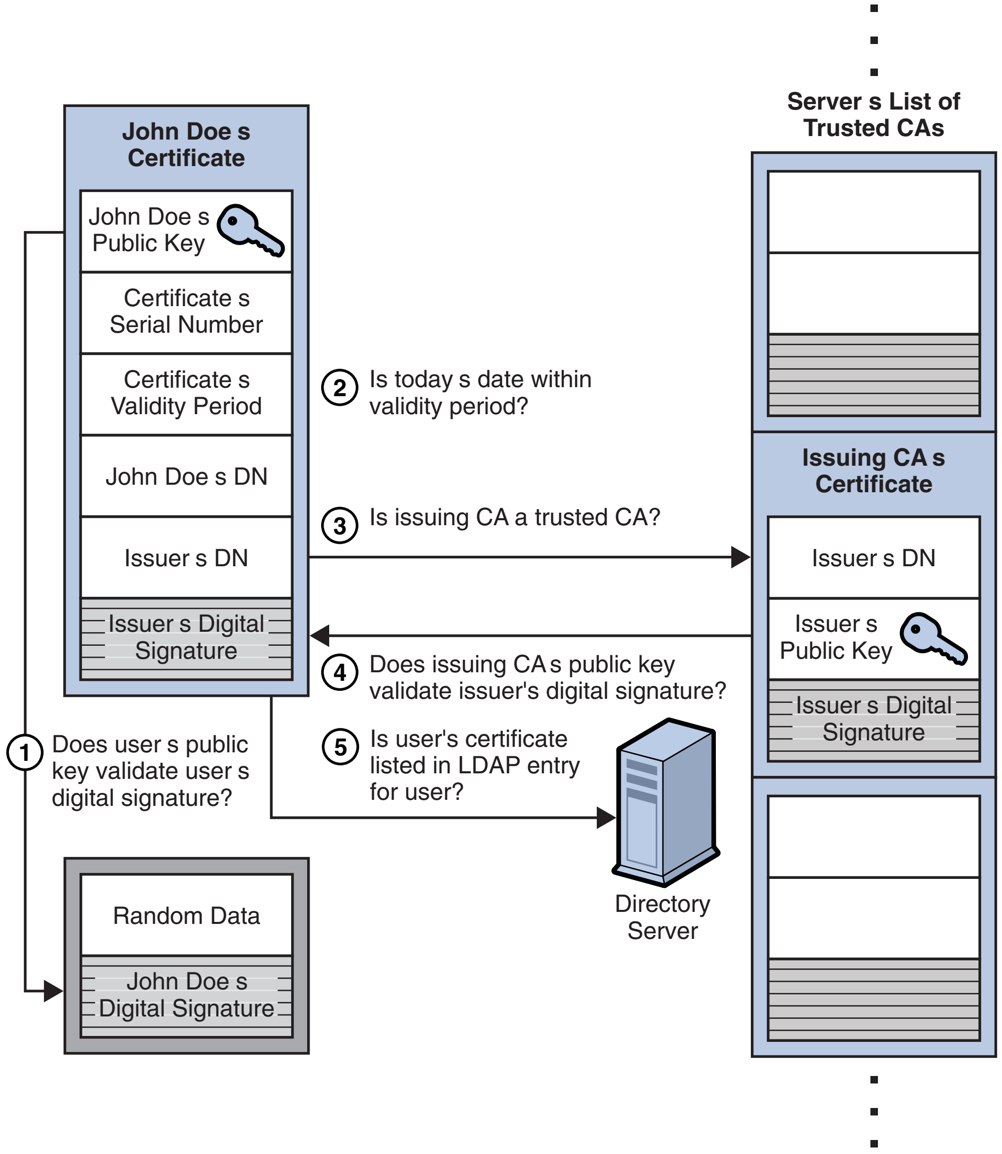

Server s List of Trusted CAs
John Doe s Certificate
John Doe s Public Key
Certificate s Serial Number
Certificate s Validity Period
John Doe s DN
Issuer s DN
Issuer s Digital Signature
2 Is today s date within validity period?
3 Is issuing CA a trusted CA?
4 Does issuing CA s public key validate issuer's digital signature?
5 Is user's certificate listed in LDAP entry for user?
1 Does user s public key validate user s digital signature?
Issuing CA s Certificate
Issuer s DN
Issuer s Public Key
Issuer s Digital Signature
Random Data
John Doe s Digital Signature
Directory Server

FIGURE 2–10 Authentication and Verification During SSL Handshake

An SSL-enabled server goes through the following steps to authenticate a user’s identity:

1. Does the user’s public key validate the user’s digital signature?

The server checks that the user's digital signature can be validated with the public key in the certificate. If so, the server has established that the public key asserted to belong to John Doe matches the private key used to create the signature and that the data has not been tampered with since it was signed.

At this point, however, the binding between the public key and the DN specified in the certificate has not yet been established. The certificate might have been created by someone attempting to impersonate the user. To validate the binding between the public key and the DN, the server must also complete steps 3 and 4 in this list.

2. Is today's date within the validity period?

   The server checks the certificate's validity period. If the current date and time are outside of that range, the authentication process won't go any further. If the current date and time are within the certificate's validity period, the server goes onto the next step.

3. Is the issuing CA a trusted CA?

   Each SSL-enabled server maintains a list of trusted CA certificates, represented by the shaded area on the right—hand side of Figure 2–10. This list determines which certificates the server accepts. If the DN of the issuing CA matches the DN of a CA on the server's list of trusted CAs, the answer to this question is yes, and the server goes on to the next step. If the issuing CA is not on the list, the client is not authenticated unless the server can verify a certificate chain ending in a CA that is trusted or not trusted within their organizations by controlling the lists of CA certificates maintained by clients and servers.

4. Does the issuing CA's public key validate the issuer's digital signature?

   The server uses the public key from the CA's certificate (which it found in its list of trusted CAs in the previous step) to validate the CA's digital signature on the certificate being presented. If the information in the certificate has changed since it was signed by the CA or if the public key in the CA certificate doesn't correspond to the private key used by the CA to sign the certificate, the server won't authenticate the user's identity. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA and proceeds. At this point, the SSL protocol allows the server to consider the client authenticated and proceed with the connection as described in step 6. The directory servers may optionally be configured to perform step 5 before step 6.

5. Is the user's certificate listed in the LDAP entry for the user?

   This optional step provides one way for a system administrator to revoke a user's certificate even if it passes the tests in all the other steps. The Certificate Management System can automatically remove a revoked certificate from the user's entry in the LDAP directory. All servers that are set up to perform this step then refuses to authenticate that certificate or establish a connection. If the user's certificate in the directory is identical to the user's certificate presented in the SSL handshake, the server goes on to the next step.

6. Is the authenticated client authorized to access the requested resources?

The server checks what resources the client is permitted to access according to the server's access control lists (ACLs) and establishes a connection with appropriate access. If the server doesn't get to step 6 for any reason, the user identified by the certificate cannot be authenticated, and the user is not allowed to access any server resources that require authentication.

Digital Signatures

Digital signatures can be used by Directory Server to maintain integrity of information. If encryption and message digests are applied to the information being sent, the recipient can determine that the information was not tampered with during transit.

Tamper detection and related authentication techniques rely on a mathematical function called a *one-way hash*. This function is also called a *message digest*. A one-way hash is a number of fixed length with the following characteristics:

- The value of the hash is unique for the hashed data. Any change in the data, even deleting or altering a single character, results in a different value.
- The content of the hashed data cannot, for all practical purposes, be deduced from the hash — which is why it is called *one-way*.

It is possible to use a private key for encryption and a public key for decryption. Although this is not desirable when you are encrypting sensitive information, it is a crucial part of digitally signing any data. Instead of encrypting the data itself, the signing software creates a one-way hash of the data, then uses your private key to encrypt the hash. The encrypted hash, along with other information, such as the hashing algorithm, is known as a digital signature. Figure 2–11 shows two items transferred to the recipient of some signed data.

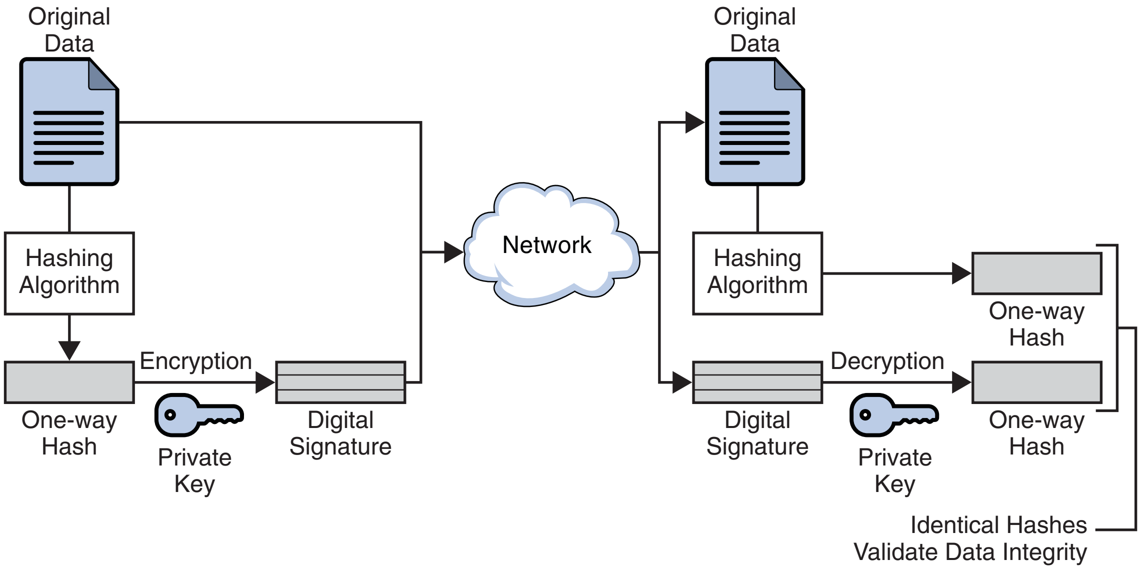


FIGURE 2–11 Digital Signatures

In Figure 2–11, the original data and the digital signature, which is basically a one-way hash (of the original data) that has been encrypted with the signer's private key. To validate the integrity of the data, the receiving software first uses the signer's public key to decrypt the hash. It then uses the same hashing algorithm that generated the original hash to generate a new one-way hash of the same data. (Information about the hashing algorithm used is sent with the digital signature, although this isn't shown in the figure.) Finally, the receiving software compares the new hash against the original hash. If the two hashes match, the data has not changed since it was signed. If they don't match, the data may have been tampered with since it was signed, or the signature may have been created with a private key that doesn't correspond to the public key presented by the signer.

If the two hashes match, the recipient can be certain that the public key used to decrypt the digital signature corresponds to the private key used to create the digital signature. Confirming the identity of the signer, however, also requires some way of confirming that the public key really belongs to a particular person or other entity.

The significance of a digital signature is comparable to the significance of a handwritten signature. Once you have signed some data, it is difficult to deny doing so later — assuming that the private key has not been compromised or out of the owner's control. This quality of digital signatures provides a high degree of nonrepudiation — that is, digital signatures make it difficult for the signer to deny having signed the data. In some situations, a digital signature may be as legally binding as a handwritten signature.

Key Encryption

With most modern cryptography, the ability to keep encrypted information secret is based not on the cryptographic algorithm, which is widely known, but on a *key*. A key is a number that must be used with the algorithm to produce an encrypted result or to decrypt previously encrypted information. For information about encryption and decryption with keys, see the following sections:

- "Symmetric-Key Encryption" on page 101
- "Public-Key Encryption" on page 102
- "Key Length and Encryption Strength" on page 103

Symmetric-Key Encryption

With symmetric-key encryption, the encryption key can be calculated from the decryption key, and vice versa. With most symmetric algorithms, the same key is used for both encryption and decryption. The following figure shows a symmetric-key encryption.

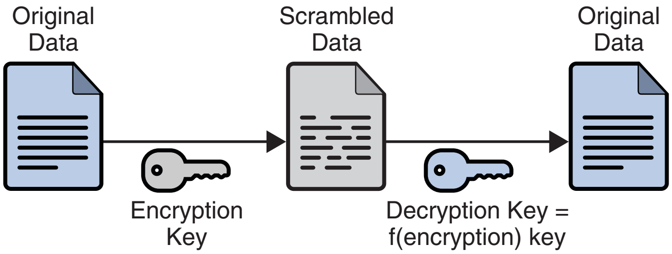


FIGURE 2–12 Symmetric-Key Encryption

Implementations of symmetric-key encryption can be highly efficient, so that users do not experience any significant time delay as a result of the encryption and decryption. Symmetric-key encryption also provides a degree of authentication, since information encrypted with one symmetric key cannot be decrypted with any other symmetric key. Thus, as long as the symmetric key is kept secret by the two parties using it to encrypt communications, each party can be sure that it is communicating with the other as long as the decrypted messages continue to make sense.

Symmetric-key encryption is effective only if the symmetric key is kept secret by the two parties involved. If anyone else discovers the key, it affects both confidentiality and authentication. A person with an unauthorized symmetric key not only can decrypt messages sent with that key, but can encrypt new messages and send them as if they came from one of the two parties who were originally using the key.

Symmetric-key encryption plays an important role in the SSL protocol, which is widely used for authentication, tamper detection, and encryption over TCP/IP networks. SSL also uses techniques of public-key encryption, which is described in the next section.

Public-Key Encryption

The most commonly used implementations of public-key encryption are based on algorithms patented by RSA Data Security. Therefore, this section describes the RSA approach to public-key encryption.

Public-key encryption (also called asymmetric encryption) involves a pair of keys—a public key and a private key—associated with an entity that needs to authenticate its identity electronically or to sign or encrypt data. Each public key is published, and the corresponding private key is kept secret. The following figure shows a simplified view of the way public-key encryption works.

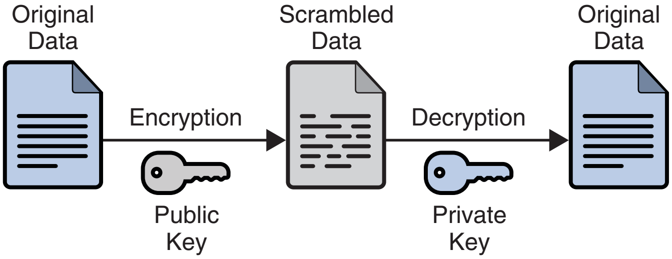


FIGURE 2–13 Public-Key Encryption

Public—key encryption lets you distribute a public key, and only you can read data encrypted by this key. In general, to send encrypted data to someone, you encrypt the data with that person's public key, and the person receiving the encrypted data decrypts it with the corresponding private key.

Compared with symmetric-key encryption, public-key encryption requires more computation and is therefore not always appropriate for large amounts of data. However, it's possible to use public-key encryption to send a symmetric key, which can then be used to encrypt additional data. This is the approach used by the SSL protocol.

As it happens, the reverse of the scheme shown in Figure 2–13 also works: data encrypted with your private key can be decrypted with your public key only. This would not be a desirable way to encrypt sensitive data, however, because it means that anyone with your public key, which is by definition published, could decrypt the data. Nevertheless, private-key encryption is useful, because it means you can use your private key to sign data with your digital signature—an important requirement for electronic commerce and other commercial applications of cryptography. Client software can then use your public key to confirm that the message was signed with your private key and that it hasn't been tampered with since being signed. "Digital Signatures" on page 99 on "Digital Signatures" on page 99 and subsequent sections describe how this confirmation process works.

Key Length and Encryption Strength

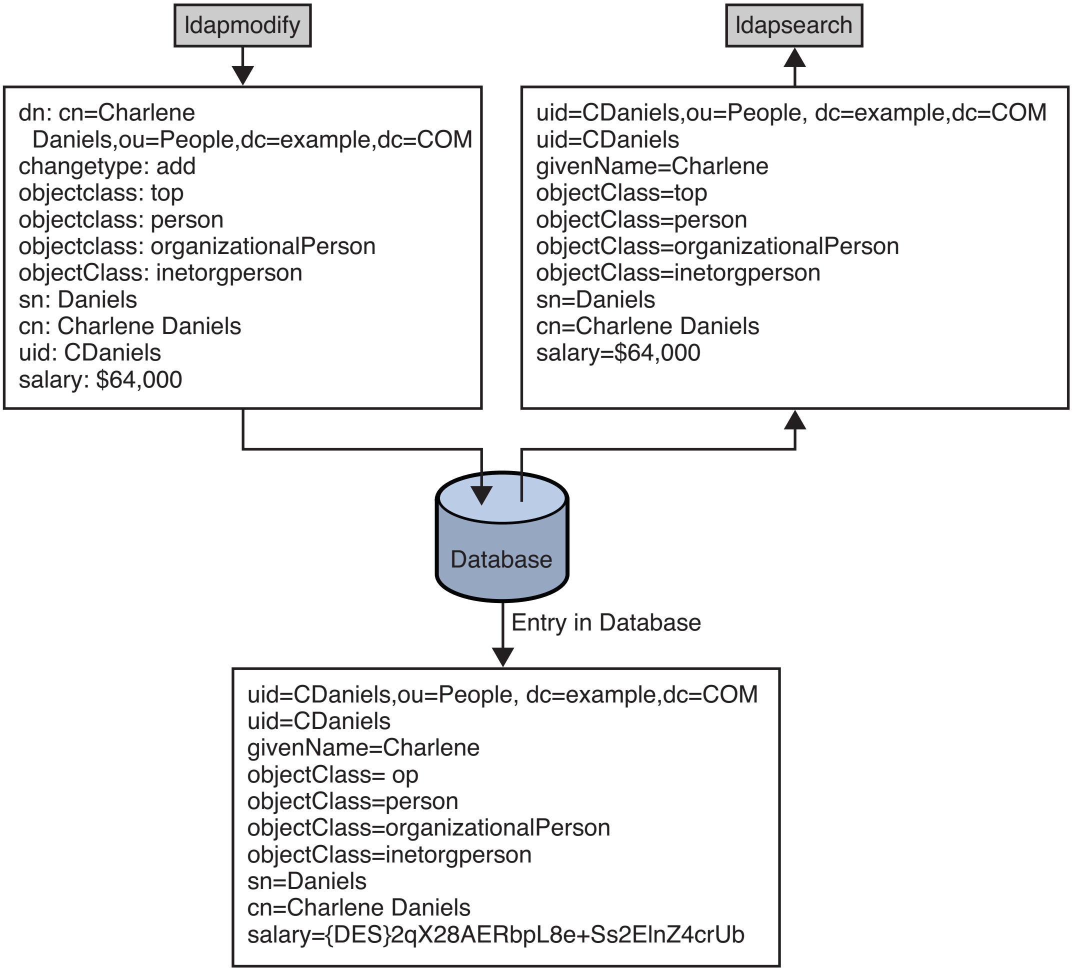
The strength of encryption is related to the difficulty of discovering the key, which in turn depends on both the cipher used and the length of the key. For example, the difficulty of discovering the key for the RSA cipher most commonly used for public-key encryption depends on the difficulty of factoring large numbers, a well-known mathematical problem.

Encryption strength is often described in terms of the size of the keys used to perform the encryption: in general, longer keys provide stronger encryption. Key length is measured in bits. For example, 128-bit keys for use with the RC4 symmetric-key cipher supported by SSL provide significantly better cryptographic protection than 40-bit keys for use with the same cipher. Roughly speaking, 128-bit RC4 encryption is 3 x 10^{26} times stronger than 40-bit RC4 encryption.

Different ciphers may require different key lengths to achieve the same level of encryption strength. The RSA cipher used for public-key encryption, for example, can use only a subset of all possible values for a key of a given length, due to the nature of the mathematical problem on which it is based. Other ciphers, such as those used for symmetric key encryption, can use all possible values for a key of a given length, rather than a subset of those values. Thus a 128-bit key for use with a symmetric-key encryption cipher would provide stronger encryption than a 128-bit key for use with the RSA public-key encryption cipher. This difference explains why the RSA public-key encryption cipher must use a 512-bit key (or longer) to be considered cryptographically strong, whereas symmetric key ciphers can achieve approximately the same level of strength with a 64-bit key. Even this level of strength may be vulnerable to attacks in the near future.

Attribute Encryption

Attribute encryption enables sensitive attributes of an entry to be stored in encrypted form. By encrypting sensitive attributes, you can prevent them from being read while the data is stored in database files, backup files, or exported LDIF files, or while the data is exported. Figure 2–14 shows a user entry being added to the database, where attribute encryption has been configured to encrypt the `salary` attribute.



```
dn: cn=Charlene
  Daniels,ou=People,dc=example,dc=COM
changetype: add
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectClass: inetorgperson
sn: Daniels
cn: Charlene Daniels
uid: CDaniels
salary: $64,000
```

```
uid=CDaniels,ou=People, dc=example,dc=COM
uid=CDaniels
givenName=Charlene
objectClass=top
objectClass=person
objectClass=organizationalPerson
objectClass=inetorgperson
sn=Daniels
cn=Charlene Daniels
salary=$64,000
```

```
uid=CDaniels,ou=People, dc=example,dc=COM
uid=CDaniels
givenName=Charlene
objectClass= op
objectClass=person
objectClass=organizationalPerson
objectClass=inetorgperson
sn=Daniels
cn=Charlene Daniels
salary={DES}2qX28AERbpL8e+Ss2ElnZ4crUb
```

FIGURE 2–14 Attribute Encryption

The attribute encryption feature supports a wide range of encryption algorithms and different platforms. Attribute encryption uses the private key of the server's SSL certificate to generate its own key. This key is then used to perform the encryption and decryption operations.

Attribute encryption is configured at the suffix level. This means that an attribute is encrypted for every entry in which it appears in a suffix. To encrypt an attribute in an entire directory, you must enable encryption for that attribute in every suffix.

If you choose to encrypt an attribute that some entries use as a naming attribute, values that appear in the DN will not be encrypted, but values stored in the entry will be encrypted.

Encrypting the `userPassword` attribute provides no security benefit unless the password needs to be stored in clear text, as is the for DIGEST-MD5 SASL authentication. If the password already has an encryption mechanism defined in the password policy, further encryption provides little additional security.

When encrypted attributes are stored, they are prefaced with a cipher tag that indicates what encryption algorithm has been used. An encrypted attribute using the DES encryption algorithm would appear as follows:

```
{CKM_DES_CBC}3hakc&jla+=snda%
```

While attribute encryption offers increased data security, the feature does impact performance. you should think carefully about which attributes require encryption and encrypt only those attributes that are particularly sensitive. Because sensitive data can be accessed directly through index files, it is necessary to encrypt the index keys corresponding to the encrypted attributes, to ensure that the attributes are fully protected.

For information about how to encrypt attributes, see "Encrypting Attribute Values" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

◆ ◆ ◆ CHAPTER 3

Directory Server Monitoring

For information about monitoring Directory Server, see the following sections.

- "Ways to Monitor Directory Server" on page 107
- "Directory Server and SNMP" on page 108
- "Directory Server and CMM/JMX" on page 110
- "Directory ServerMonitoring Attributes" on page 111

Ways to Monitor Directory Server

Directory Server can be monitored in the following ways:

Directory Service Control Center

Directory Service Control Center, DSCC, can be used to monitor current activities of a Directory Server instance.

DSCC provides general server information, including a resource summary, current resource usage, connection status, and global database cache information. It also provides general database information, such as the database type, status, and entry cache statistics. Cache information and information relative to each index file within the database is also provided. In addition, DSCC provides information relative to the connections and the operations performed on each chained suffix.

Command line

The `dsconf` command can be used to configure logging and to monitor the replication status of Directory Server. For information about how to configure logging, see "Configuring Logs for Directory Server" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*. For information about how to use the `dsconf` command for monitoring, see "Getting Replication Status by Using the Command Line" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

The ldapsearch command can be used to search the cn=monitor entry for information about current activities of a Directory Server instance. For information about cn=monitor, see "Directory ServerMonitoring Attributes" on page 111.

Log analyzer tool
The Directory Server Resource Kit provides a log analyzer tool called logconv(1).

The logconv tool extracts usage statistics and counts the occurrences of significant events in the access logs.

Java Management Extensions, JMX
Directory Server exposes management information through JMX according to the Common Monitoring Information and Data Model. See the *Sun Java Enterprise System 5 Monitoring Guide* for details.

Java ES Monitoring Framework, Java ES MF, provides an JMX entry point to retrieve data. For information about the JMX entry points exposed for monitoring Directory Server, see "Directory Server and SNMP" on page 108.

Simple Network Management Protocol, SNMP
Directory Server exposes management information through SNMP. See the *Sun Java Enterprise System 5 Monitoring Guide* for details.

Java ES MF provides an SNMP entry point to retrieve SNMP data. For information about the SNMP entry points exposed for monitoring Directory Server, see "Directory Server and SNMP" on page 108.

Simple Object Access Protocol, SOAP
Java ES MF provides a SOAP entry point to retrieve data. See the *Sun Java Enterprise System 5 Monitoring Guide* for details.

Directory Server and SNMP

Directory Server implements the dsTable and the dsApplIfOpsTable of the Directory Server Monitoring MIB defined by RFC 2605 (http://www.ietf.org/rfc/rfc2605.txt). It does not implement the dsIntTable.

Directory Server also implements the Network Services Monitoring MIB defined by RFC 2788 (http://www.ietf.org/rfc/rfc2788.txt).

Directory Server support for SNMP has the following limitations.

- SNMP support is for monitoring only, no SNMP management is supported.
- No SNMP traps are implemented.

This rest of this section explains how the information flows from the monitoring application to Directory Server and back, particularly in the case where you use SNMP.

The SNMP interface is exposed by Java ES MF. See the *Sun Java Enterprise System 5 Monitoring Guide* for details.

The monitoring framework is contained within the Common Agent Container, cacao, which is installed alongside Directory Server. Figure 3–1 shows the monitoring framework.

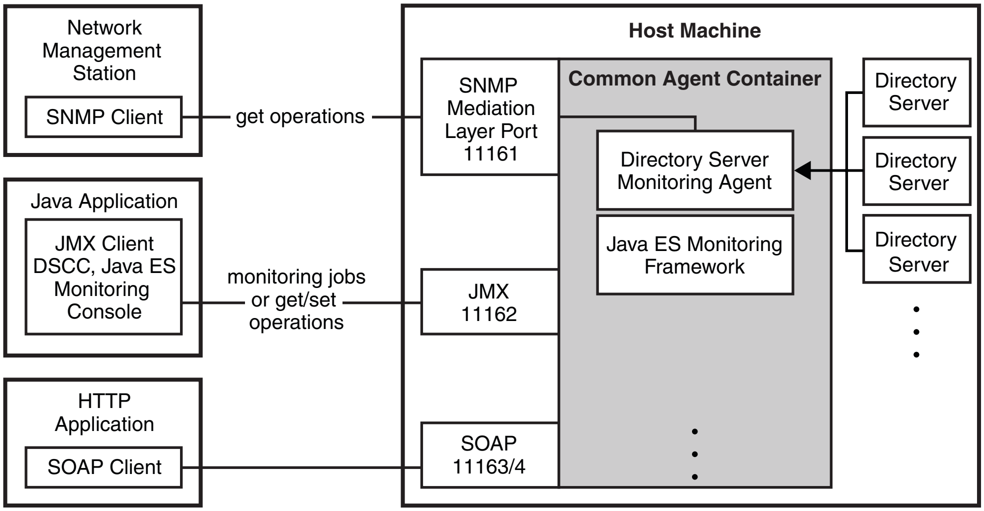

Network Management Station
SNMP Client
get operations
Java Application
JMX Client DSCC, Java ES Monitoring Console
monitoring jobs or get/set operations
HTTP Application
SOAP Client
Host Machine
SNMP Mediation Layer Port 11161
JMX 11162
SOAP 11163/4
Common Agent Container
Directory Server Monitoring Agent
Java ES Monitoring Framework
Directory Server
Directory Server
Directory Server

FIGURE 3–1 Overall Monitoring Information Flow

SNMP support for monitoring Directory Server is managed by a Directory Server agent in the Common Agent Container. On Directory Server startup, the Monitoring server plug-in registers theDirectory Server instance with the Directory Server agent within the Common Agent Container.

Figure 3–2 shows how SNMP information about Directory Server flows through the Common Agent Container.

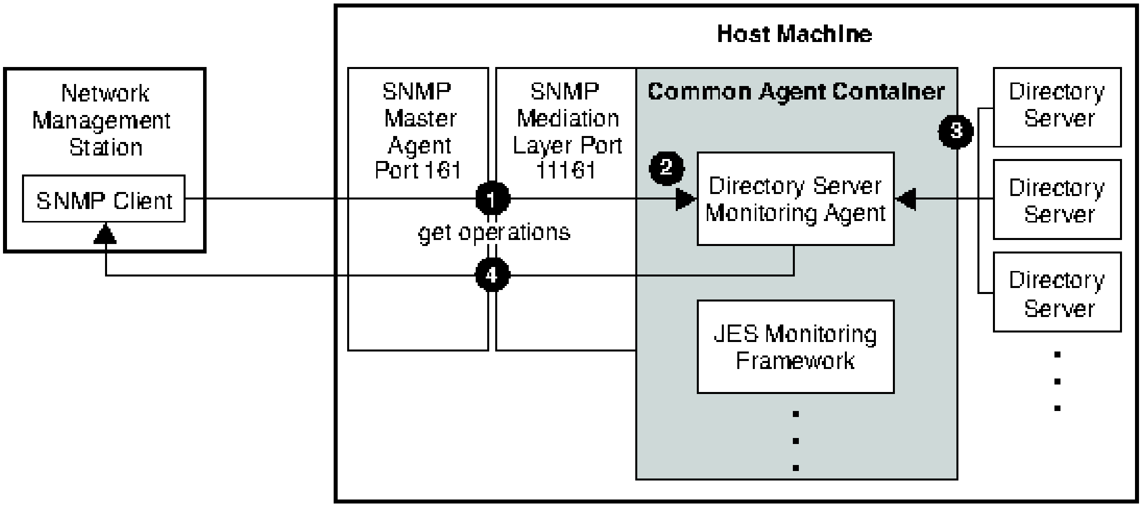


FIGURE 3–2 SNMP Information Flow

SNMP information about Directory Server flows as follows.

1. The network management station sends a GET message through the master SNMP agent, which by default uses standard port 161, to the SNMP mediation layer, which by default uses port 11161.

   For information about how to configure access to the SNMP mediation layer, see the *Sun Java Enterprise System 5 Monitoring Guide*.
2. The SNMP mediation layer forwards any requests destined for the Directory Server to the Directory Server agent.
3. When the server state changes, Directory Server pushes SNMP information to the Directory Server agent.
4. The Directory Server agent relays the response back to the SNMP client via the SNMP mediation layer and master SNMP agent to the network management station. The network management station then displays the data through its network management application.

Directory Server and CMM/JMX

Directory Server supports monitoring through JMX, which is exposed by Java ES MF. See the *Sun Java Enterprise System 5 Monitoring Guide* for details on the interface itself.

The monitoring framework is contained within the Common Agent Container, cacao, which is installed alongside Directory Server. Figure 3–1 shows the monitoring framework. The information flow for JMX is similar to the flow shown for SNMP in Figure 3–2.

The monitoring information exposed through JMX is organized according to the Common Monitoring Information and Data Model, CMM. CMM allows applications exposing their monitoring information to associate human-readable descriptions with the individual counters and other information. CMM is therefore meant to be self-documenting. Directory Server implements the following CMM classes.

- CMM_ApplicationSystem
- CMM_ApplicationSystemSetting
- CMM_ApplicationSystemStats
- CMM_InstalledProduct
- CMM_RFC2605ApplicationSystemSettings
- CMM_RFC2605ApplicationSystemStats
- CMM_RFC2605ServiceAccessURIStats
- CMM_ServiceAccessBySAP
- CMM_ServiceAccessBySAPStats
- CMM_ServiceAccessURI
- CMM_ServiceAccessURIStats

When examining the content of the monitoring information, notice that CMM_ServiceAccessURI is implemented not only for LDAP and for LDAPS, but also for DSML/HTTP or DSML/HTTPS if the DSML front end has been enabled.

Java ES Monitoring Console offers a browser-based user interface to examine the information exposed. See the *Sun Java Enterprise System 5 Monitoring Guide* for instructions on preparing the Monitoring Console for use.

Directory ServerMonitoring Attributes

Read-only monitoring information is stored under the cn=monitor entry.

cn=monitor

The cn=monitor entry is an instance of the extensibleObject object class. For cn=monitor configuration attributes to be taken into account by the server, this object class, in addition to the top object class, is present in the entry. The cn=monitor read-only attributes are presented in this section.

backendMonitorDN

DN for each Directory Server backend.

For further database monitoring information, refer to dse.ldif(4).

bytesSent

Number of bytes sent by Directory Server.

cache-avail-bytes

The number of bytes available for caching.

connection

List of open connections given in the following format:

```
connection=31:20010201164808Z:45:45::cn=admin,cn=Administrators,cn=config:LDAP
```

- `31` is number of the file descriptor used by the server in handling the connection
- `20010201164808Z` is the date the connection was opened
- `45` is the number of operations received
- `45` is the number of completed operations
- `cn=admin,cn=Administrators,cn=config` is the bind DN

connectionPeak

Maximum number of simultaneous connections since server startup.

currentConnections

Number of current Directory Server connections.

currentTime

Current time usually given in Greenwich Mean Time, indicated by GeneralizedTime syntax `Z` notation, for example `20010202131102Z`.

dTableSize

Size of the Directory Server descriptor table.

entriesSent

Number of entries sent by Directory Server.

nbackEnds

Number of Directory Server backends.

opsCompleted

Number of Directory Server operations completed.

opsInitiated

Number of Directory Server operations initiated.

request-que-backlog

The number of requests waiting to be processed by a thread. Each request received by the server is accepted, then placed in a queue until a thread is available to process it. The queue backlog should always be small, `0` or close to `0`. If the queue backlog is large, use the `nsslapd-threadnumber` attribute to increase the number of threads available in the server.

readWaiters

Number of connections where some requests are pending and not currently being serviced by a thread in Directory Server.

currentpsearches

Number of persistent searches currently running on the server. You can set a maximum number of persistent searches on the server by using the command `dsconf set-server-prop max-psearch-count:`*number*.

startTime

Directory Server start time.

threads

Number of operation threads Directory Server creates during startup. This attribute can be set using the `nsslapd-threadnumber` attribute under `cn=config`. The `nsslapd-threadnumber` attribute is not present in the configuration by default, but can be added.

totalConnections

Total number of Directory Server connections.

version

Directory Server version and build number.

cn=disk,cn=monitor

The `cn=disk` entry enables you to monitor disk conditions over LDAP. This entry is an instance of the `extensibleObject` object class. A `cn=`*disknumber*`,cn=disk,cn=monitor` entry exists for each disk. The following disk monitoring attributes appear under each of these individual disk entries.

disk-dir

Specifies the pathname of a directory used by the server on disk. Where several database instances reside on the same disk or an instance refers to several directories on the same disk, the short pathname is displayed. The disk numbering is arbitrary.

disk-free

Indicates the amount of free disk space available to the server, in MB.

Note – The disk space available to the server process may be less than the total free disk space. For example, on some platforms a process that is not running as `root` may not have all the free disk space available to it.

disk-state

Indicates the state of the disk, based on the available free space and on the thresholds set for disk low and disk full with the configuration parameters `nsslapd-disk-low-threshold` and `nsslapd-disk-full-threshold`. Possible values are `normal`, `low`, and `full`.

cn=counters,cn=monitor

This entry holds counter information for the various subtree entry counter plug-ins, if they are enabled.

cn=monitor,cn=Class of Service,cn=plugins, cn=config

This entry holds counters related to the Class of Service plug-in. This entry is an instance of the `extensibleObject` object class.

classicHashAvgClashListLength

When the CoS plug-in uses the hash table for fast lookup, if more than one classic CoS template corresponds to the hash key used, the plug-in next checks for matches in what is called the clash list, a list of templates sharing an identical hash key. The value of this attribute provides the average length across all hash tables of classic CoS template clash lists, giving some indication of how much linear searching the plug-in must perform after using the hash table during fast lookup.

classicHashAvgClashPercentagePerHash

The average number of clashes per hash table. That is, the average percentage per hash of classic CoS templates sharing an identical hash key.

classicHashMemUsage

The memory overhead in bytes to hold hash tables for fast classic CoS template lookups.

classicHashValuesMemUsage

The memory in bytes used to hold hash values for fast classic CoS template lookups.

numClassicDefinitions

The number of classic CoS definition entries in use.

numClassicHashTables

The number of hash tables created for fast lookup where more than 10 classic CoS templates apply for a single CoS definition. Hash tables are not created for smaller lists of templates.

numClassicTemplates

The number of classic CoS template entries in use.

numCoSAttributeTypes

The number of distinct attributes with values calculated through CoS.

numIndirectDefinitions

The number of indirect CoS definition entries in use.

numPointerDefinitions

The number of pointer CoS definition entries in use.

numPointerTemplates

The number of pointer CoS template entries in use.

CHAPTER 4

Directory Server Replication

This chapter includes the following sections:

Introduction to Replication

This introduction to replication addresses the following topics:

Types of Replica

A database that participates in replication is called a *replica*. There are three kinds of replica:

- A *master* replica is a read-write database that contains a master copy of the directory data. A master replica can perform the following tasks:
  - Respond to update requests and read requests from directory clients
  - Maintain historical information and a change log for the replica

  - Initiate replication to consumers or hubs
- A *consumer* replica is a read-only database that contains a copy of the information held in a master replica. A consumer replica can perform the following tasks:
  - Respond to read requests
  - Maintain historical information for the replica
  - Refer update requests to servers that contain a master replica
- A *hub* replica is a read-only database, like a consumer replica, but stored on a directory server that supplies one or more consumer replicas. A hub replica can perform the following tasks:
  - Respond to read requests
  - Maintain historical information and a change log for the replica
  - Initiate replication to consumers
  - Refer update requests to servers that contain a master replica

A single instance of Directory Server can be configured to manage several replicas.

A replica can act as a supplier of updates, or a consumer of updates, or both.

- A *supplier* is a replica that copies information to another replica.

  A master replica can be a supplier to a hub replica and a consumer replica. A hub replica can be a supplier to a consumer replica. In multi-master replication, one master replica can be a supplier to another master replica.
- A *consumer* is a replica to which another replica copies information.

  A hub replica and a consumer replica can be consumers of a master replica. A consumer replica can be a consumer of a hub replica. In multi-master replication, one master replica can be a consumer of another master replica.

A replica can be *promoted* or *demoted* to change its behavior with respect to other replicas. Dedicated consumers can be promoted to hubs, and hubs can be promoted to masters. Masters can be demoted to hubs, and hubs can be demoted to dedicated consumers.

A server that contains a consumer replica only is called a *dedicated consumer*.

Unit of Replication

The smallest logical unit of replication is a *suffix*, also known as a naming context. The term suffix arises from the way the base DN for the naming context is a suffix for all DNs in that context. For example, the suffix `dc=example,dc=com` contains all directory entries in the Example.com naming context.

The replication mechanism requires one suffix to correspond to one database. The unit of replication applies to both suppliers and consumers. Therefore, two suffixes on a master replica cannot be replicated to one suffix on a consumer replica, and vice versa.

Replica Identity

Master replicas require a unique replica identifier that is a 16-bit integer between 1 and 65534. Consumer and hub replicas all have the replica ID of 65535. The replica ID identifies the replica on which changes are made.

If multiple suffixes are configured on one master, you can use the same replica ID for each suffix on the master. In this way, when a change is made on that replica ID, it is possible to identify the server on which change was made.

Replication Agreements

Replication agreements define the relationships between a supplier and a consumer. The replication agreement is configured on the supplier. A replication agreement contains the following replication parameters:

- The suffix to replicate.
- The consumer server to which the data is pushed.
- The replication schedule.
- The bind DN and credentials the master must use to bind to the consumer.
- How the connection is secured.
- Which attributes to exclude or include in fractional replication, if fractional replication is configured.
- The group and window sizes to configure the number of changes you can group into one request and the number of requests that can be sent before consumer acknowledgement is required.
- Information about the replication status for this agreement.
- The level of compression used in replication on Solaris and Linux systems.

Replication Authentication

Before a master can update a consumer, the consumer authenticates the master by using a special entry called the *Replication Manager entry*. The master uses the Replication Manager entry to bind to the consumer.

The Replication Manager entry has a special user profile that bypasses all access control rules defined on the consumer server. The special user profile is only valid in the context of replication.

The Replication Manager entry has the following characteristics.

- On a consumer server, the Replication Manager is the user who is allowed to perform updates. The entry for Replication Manager must be present for all replicas.
- The bind DN of the Replication Manager entry is set in the replication agreement. The bind DN must be configured for hubs, or masters to point to an existing Replication Manager entry.
- For initialization and security reasons, the Replication Manager entry cannot be part of the replicated data.

The Replication Manager entry is created by default when you configure replication through the browser-based interfaceDirectory Service Control Center. You can also create your own Replication Manager entry. For information about how to create a Replication Manager entry, see "Using a Non-Default Replication Manager" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Authentication can be performed in the following ways for SSL with replication.

- For SSL server authentication, you must have a Replication Manager entry, and its associated password, in the server you are authenticating to.
- For SSL client authentication, you must have an entry that contains a certificate in the server you are authenticating to. This entry may or may not be mapped to the Replication Manager entry.

Replication Change Log

All modifications received by a master replica are recorded in a change log. A change log is maintained on all master replicas and hub replicas.

In earlier versions of Directory Server, the change log was accessible over LDAP. In this version of the product, the change log is not accessible over LDAP but is stored in its own database. If your application needs to read the change log, use the retro change log plug-in for backward compatibility. For more information about the retro change log plug-in, see "Replication and the Retro Change Log Plug-In" on page 130.

Change Sequence Number

Each change to a master replica is identified by a change sequence number, CSN. The CSN is generated by the master server and is not visible to the client application. The CSN contains the timestamp, a sequence number, the replica ID, and a subsequence number. The change log is ordered by the CSN.

Replica Update Vector

The replica update vector, RUV, identifies the state of each replica in a topology. Stored on the supplier and on the consumer, the RUV is used to establish which changes need to be replicated. The RUV stores the URL of the supplier, the ID of the supplier, the minimum CSN, and the maximum CSN.

RUVs can be read through `nsds50ruv`(5dsconf) and `nsds50ruv`(5dsconf) attributes.

Deleted Entries: Tombstones

Directory entries deleted on one replica are maintained by Directory Server until no longer needed for replication. Such deleted entries are called tombstones, as they have `objectclass: nsTombstone`. In rare cases, you might need to remove tombstones manually over LDAP.

Tombstones visible only to Directory Manager. Furthermore, tombstones show up only in a search with filter `(objectclass=nsTombstone)`. The following `ldapsearch` command returns tombstone entries under `dc=example,dc=com`.

```
$ ldapsearch -D "cn=Directory Manager" -b dc=example,dc=com "(objectclass=nsTombstone)"
```

Consumer Initialization and Incremental Updates

During consumer initialization, or total update, all data is physically copied from a master to a consumer. When you have created a replication agreement, the consumer defined by that agreement must be initialized. When a consumer has been initialized, the master can begin to replay, or replicate, update operations to the consumer. Under normal circumstances, the consumer should not require further initialization. However, if the data on a master is restored from a backup, it might be necessary to re-initialize the consumers that depend on that master.

In a multi-master replication topology, the default behavior of a read-write replica that has been re-initialized from a backup or from an LDIF file, is to refuse client update requests. By default, the replica remains in read-only mode until it is configured to accept updates again. You set the suffix property `repl-accept-client-update-enabled` to `on` using the `dsconf set-suffix-prop` command when the oldest updates are on the read-only replica.

When a consumer has been initialized, replication updates are sent to the consumer when the modifications are made on the supplier. These updates are called *incremental updates*. A consumer can be incrementally updated by several suppliers at once, provided that the updates originate from different replica IDs.

The *binary copy* feature can be used to clone master replicas or consumer replicas by using the binary backup files of one server to restore another server. For information about how to use binary copy for replication, see “Initializing a Replicated Suffix by Using Binary Copy” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Referrals and Replication

When a consumer receives a request to modify data, it *does not* forward the request to the server that contains the master replica. Instead, it returns to the client a list of the URLs of the masters that can satisfy the request. These URLs are called referrals.

The replication mechanism automatically configures consumers to return referrals for all known masters in the replication topology. However, you can also add your own referrals and overwrite the referrals set automatically by the server. The ability to control referrals helps enables you to perform the following tasks:

- Point referrals to secure ports only
- Point to a Directory Proxy Server instead for load balancing
- Redirect to local servers only in the case of servers separated by a WAN
- Limit referrals to a subset of masters in four-way multi-master topologies

Directory Proxy Server is able to follow referrals.

Replication Configurations

This section covers the following topics:

- “Multi-Master Replication” on page 122
- “Cascading Replication” on page 127
- “Prioritized Replication” on page 128
- “Fractional Replication” on page 129

For information about planning your replication, see the *Sun Java System Directory Server Enterprise Edition 6.3 Deployment Planning Guide*.

Multi-Master Replication

In multi-master replication, replicas of the same data exist on more than one server. For information about multi-master replication, see the following sections:

- “Concepts of Multi-Master Replication” on page 122
- “Multi-Master Replication Over Wide Area Networks” on page 123
- “Fully Meshed Multi-Master Topology” on page 124

Concepts of Multi-Master Replication

In a multi-master configuration, data is updated on multiple masters. Each master maintains a change log, and the changes made on each master are replicated to the other servers. Each master plays the role of supplier and consumer.

Multi-master replication can cause synchronization conflicts. Conflicts are usually resolved automatically by using the timestamp associated with each change, where the most recent change takes precedence. Some rare conflicts must be resolved manually. For more information, see “Solving Common Replication Conflicts” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Each supplier in a multi-master environment must have a replication agreement. The following figure shows two master servers and their replication agreements.

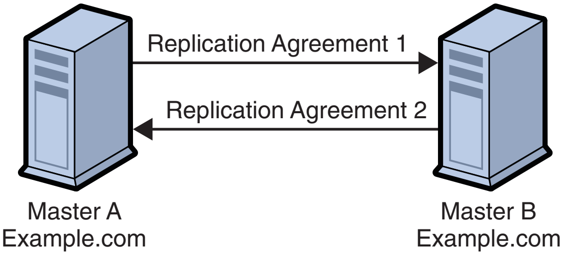


FIGURE 4–1 Multi-Master Replication Configuration (Two Masters)

In the preceding figure, Master A and Master B have a master replica of the same data. Each master has a replication agreement that specifies the replication flow. Master A acts as a master in the scope of Replication Agreement 1, and as a consumer in the scope of Replication Agreement 2.

Multi-master replication can be used for the following tasks:

- To replicate updates by using the replica ID.

  Updates by using the replica ID make it possible for a consumer to be updated by multiple suppliers at the same time, provided that the updates originate from different replica IDs.
- To enable or disable a replication agreement.

  Replication agreements can be configured but left disabled, then enabled rapidly when required. This feature provides flexibility in replication configuration. This can be done whether you use multiple masters or not.

Multi-Master Replication Over Wide Area Networks

Directory Server supports multi-master replication over WANs, enabling multi-master replication configurations across geographical boundaries in international, multiple data center deployments.

The replication protocol provides full asynchronous support, window and grouping mechanisms, and support for compression. These features render multi-master replication over WAN a viable deployment possibility.

In a multi-master replication over WAN configuration, *all* instances of Directory Server separated by a WAN *must* support multi-master replication over WANs.

Group Mechanism and Window Mechanism

The group mechanism and window mechanism can be used to group changes rather than send them individually. The group mechanism and window mechanism can also be used to specify a number of requests that can be sent to the consumer without the supplier waiting for an acknowledgement from the consumer.

For information about how to adjust the group size and window size, see "Configuring Network Parameters" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Replication Compression Mechanisms

Replication compression helps to streamline replication flow and avoid bottlenecks caused by limited bandwidth.

Fully Meshed Multi-Master Topology

In a *fully meshed* multi-master topology, each master is connected to each of the other masters. A fully meshed topology provides high availability and guaranteed data integrity. The following figure shows a fully meshed, four-way, multi-master replication topology with some consumers.

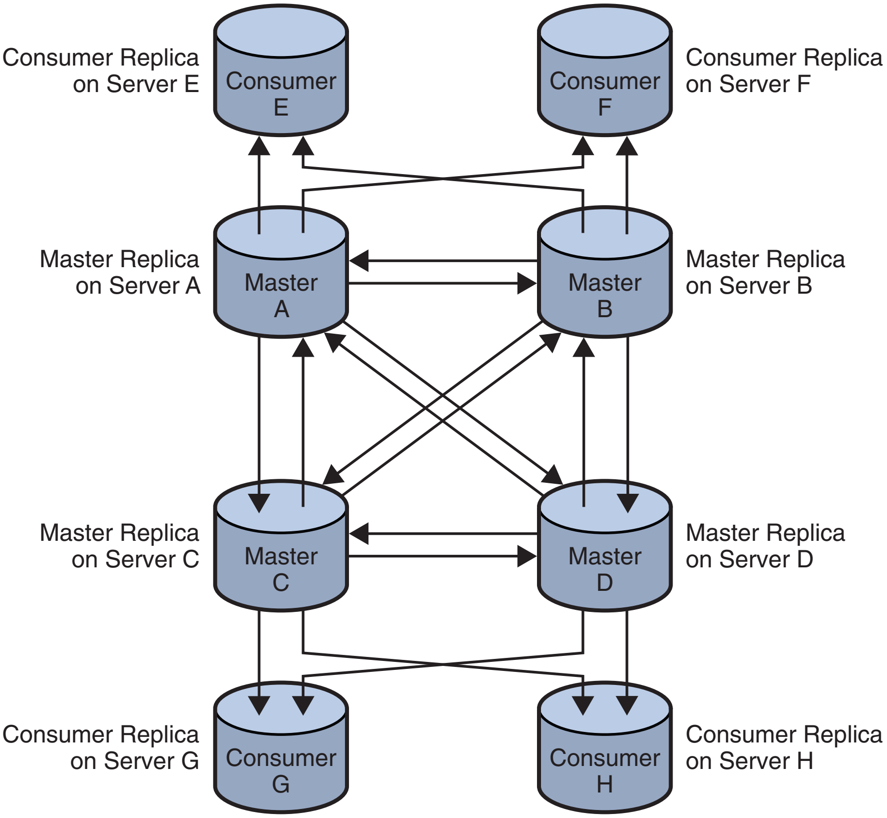

Consumer Replica on Server E
Consumer E
Consumer F
Consumer Replica on Server F
Master Replica on Server A
Master A
Master B
Master Replica on Server B
Master Replica on Server C
Master C
Master D
Master Replica on Server D
Consumer Replica on Server G
Consumer G
Consumer H
Consumer Replica on Server H

FIGURE 4–2 Fully Meshed, Four-Way, Multi-Master Replication Configuration

In Figure 4–2, the suffix is held on four masters to ensure that it is always available for modification requests. Each master maintains its own change log. When one of the masters processes a modification request from a client, it records the operation in its change log. The master then sends the replication update to the other masters, and in turn to the other consumers. Each master also stores a Replication Manager entry used to authenticate the other masters when they bind to send replication updates.

Each consumer stores one or more entries that correspond to the Replication Manager entries. The consumers use the entries to authenticate the masters when they bind to send replication updates. It is possible for each consumer to have just one Replication Manager entry that enables all masters to use the same Replication Manager entry for authentication. By default, the consumers have referrals set up for all masters in the topology. When consumers receive modification requests from the clients, they send the referrals to back to the client. For more information about referrals, see "Referrals and Replication" on page 122.

Figure 4–3 presents a detailed view of the replication agreements, change logs, and Replication Manager entries that must be set up on Master A.Figure 4–4 provides the same detailed view for Consumer E.

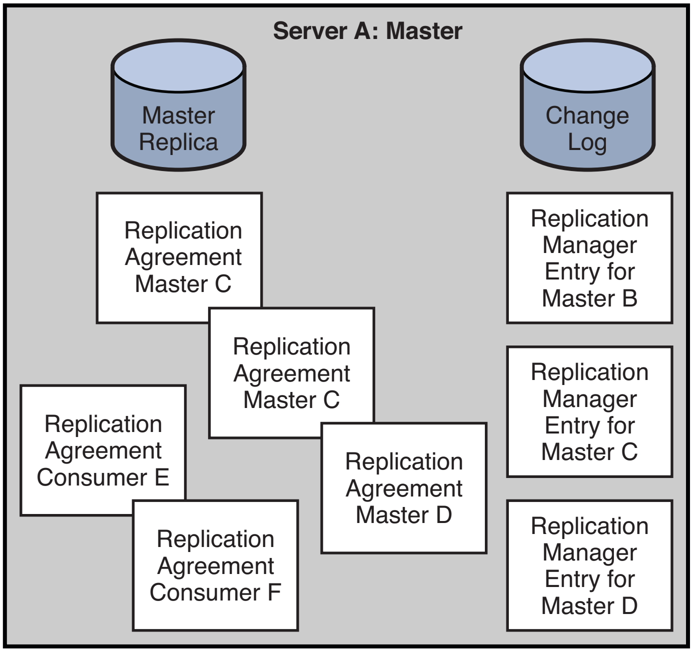

Server A: Master
Master Replica
Change Log
Replication Agreement Master C
Replication Manager Entry for Master B
Replication Agreement Master C
Replication Manager Entry for Master C
Replication Agreement Consumer E
Replication Agreement Master D
Replication Agreement Consumer F
Replication Manager Entry for Master D

FIGURE 4–3 Replication Configuration for Master A (Fully Meshed Topology)

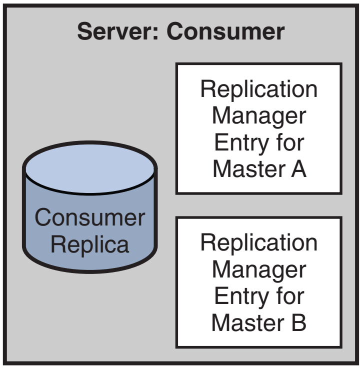

Server: Consumer
Replication Manager Entry for Master A
Consumer Replica
Replication Manager Entry for Master B

FIGURE 4–4 Replication Configuration for Consumer Server E (Fully Meshed Topology)

Master A requires the following:

- A master replica
- A change log

- Replication Manager entries for Masters B, C, and D, unless you use the same Replication Manager entry on each replica
- Replication agreements for Masters B, C, and D, and for Consumers E, and F

Consumer E requires the following:

- A consumer replica
- Replication Manager entries to authenticate Masters A, and B when they bind to send replication updates

Cascading Replication

In a cascading replication configuration, a server acting as a hub receives updates from a server acting as a supplier. The hub replays those updates to consumers. The following figure illustrates a cascading replication configuration.

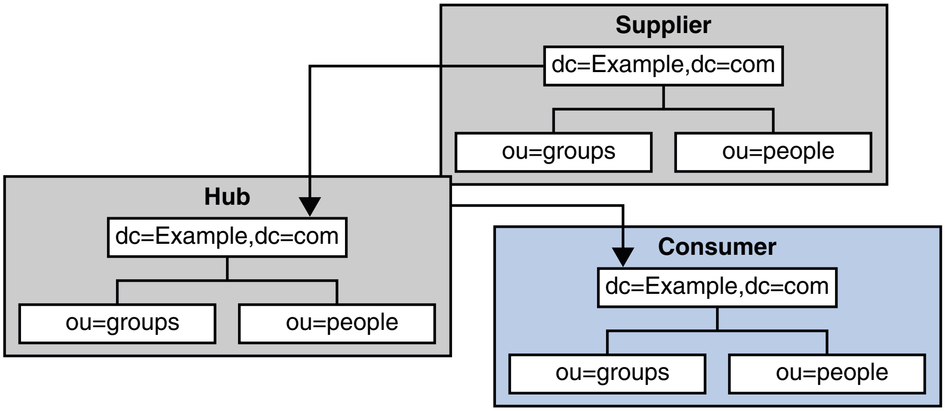


FIGURE 4–5 Cascading Replication Configuration

Cascading replication is useful in the following scenarios:

- When there are a lot of consumers.

  Because the masters in a replication topology handle all update traffic, it could put them under a heavy load to support replication traffic to the consumers. You can off-load replication traffic to several hubs that can each service replication updates to a subset of the consumers.
- To reduce connection costs by using a local hub in geographically distributed environments.

The following figure shows cascading replication to a large number of consumers.

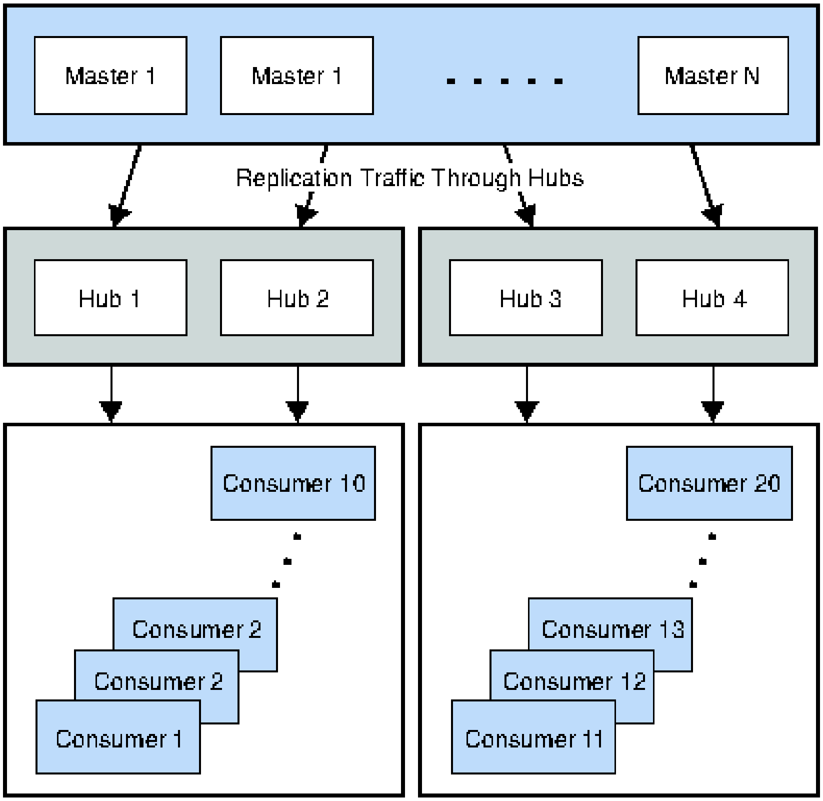


FIGURE 4–6 Cascading Replication to a Large Number of Consumers

In Figure 4–6, hubs 1 and 2 relay replication updates to consumers 1 through 10, leaving the master replicas with more resources to process directory updates.

The masters and the hubs maintain a change log. However, only the masters can process directory modification requests from clients. The hubs contains a Replication Manager entry for each master that sends updates to them. Consumers 1 through 10 contain Replication Manager entries for hubs 1 and 2.

The consumers and hubs can process search requests received from clients, but cannot process modification requests. The consumers and hubs refer modification requests to the masters.

Prioritized Replication

In previous versions of Directory Server, updates were replicated in chronological order. In this version of the product, updates can be prioritized for replication. Priority is a boolean feature, it is on or off. There are no levels of priority. In a queue of updates waiting to be replicated, updates with priority are replicated before updates without priority.

Priority rules are configured with the following replication priority rule properties:

- The identity of the client, `bind-dn`.
- The type of update, `op-tyupe`.
- The entry or subtree that was updated, `base-dn`.
- The attributes changed by the update, `att`.

For information about these properties, see `repl-priority`(5dsconf).

When the master replicates an update to one or more hubs or consumer replicas, the priority of the update is the same across all of the hubs and consumer replicas. If one parameter is configured in a priority rule for prioritized replication, all updates that match that parameter are prioritized for replication. If two or more parameters are configured in a priority rule for prioritized replication, all updates that match *all* parameters are prioritized for replication.

In the following scenario, it is possible that a master replica attempts to replicate an update to an entry before it has replicated the addition of the entry:

- The entry is added on the master replica and then updated on the master replica
- The update operation has replication priority but the add operation does not have replication priority

In this scenario, the update operation cannot be replicated until the add operation is replicated. The update waits for its chronological turn, after the add operation, to be replicated.

Fractional Replication

Fractional replication can be used to replicate a subset of the attributes of all entries in a suffix or sub-suffix. Fractional replication can be configured, per agreement, to *include* attributes in the replication or to *exclude* attributes from the replication. Usually, fractional replication is configured to exclude attributes. The interdependency between features and attributes make managing a list of included attributes difficult.

Fractional replication can be used for the following purposes:

- To filter content for synchronization between intranet and extranet servers
- To reduce replication costs when a deployment requires only certain attributes to be available everywhere

Fractional replication is configured with the replication agreement properties `repl-fractional-include-attr` and `repl-fractional-exclude-attr` attributes. For information about these properties, see `repl-agmt`(5dsconf). For information about how to configure fractional replication, see “Fractional Replication” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Fractional replication is *not* backward compatible with versions of Directory Server prior to Directory Server 5.2. If you are using fractional replication, ensure that *no* other instances of Directory Server are prior to Directory Server 5.2.

Replication and the Retro Change Log Plug-In

The retro change log is a plug-in used by LDAP clients for maintaining application compatibility with earlier versions of Directory Server. The retro change log is stored in a separate database from the Directory Server change log, under the suffix `cn=changelog`.

A retro change log can be enabled on a standalone server or on each server in a replication topology. When the retro change log is enabled on a server, updates to all suffixes on that server are logged by default.

For information about how to use the retro change log, see "Using the Retro Change Log" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Retro Change Log and Multi-Master Replication

When a retro change log is enabled with replication, the retro change log receives updates from all master replicas in the topology. The updates from each master replica are combined in the retro change log. The following figure illustrates the retro change log on two servers in a multi-master topology.

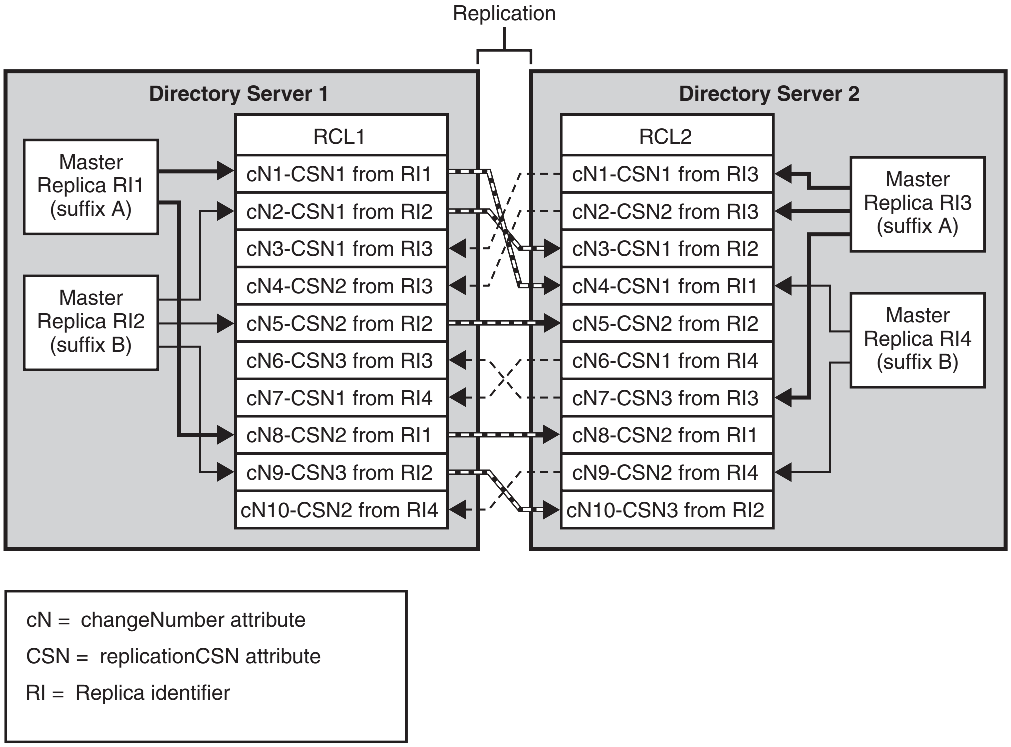


FIGURE 4–7 Retro Change Log and Multi-Master Replication

The retro change log uses the following attributes during replication:

`changeNumber` (`cN`) Identifies the order in which an update is logged to the retro change log

`replicationCSN` (`CSN`) Identifies the time when an update is made to a given replica

`replicaIdentifier` (`RI`) Identifies the replica that is updating the retro change log

The diagram shows that the retro change logs, RCL1 and RCL2, contain the same list of updates, but that the updates do not have the same order. However, for a given `replicaIdentifier`, updates are logged in the same order on each retro change log. The order in which updates are logged to the retro change log is given by the `changeNumber` attribute.

Failover of the Retro Change Log

The following figure illustrates a simplified replication topology where a client reads a retro change log on a consumer server.

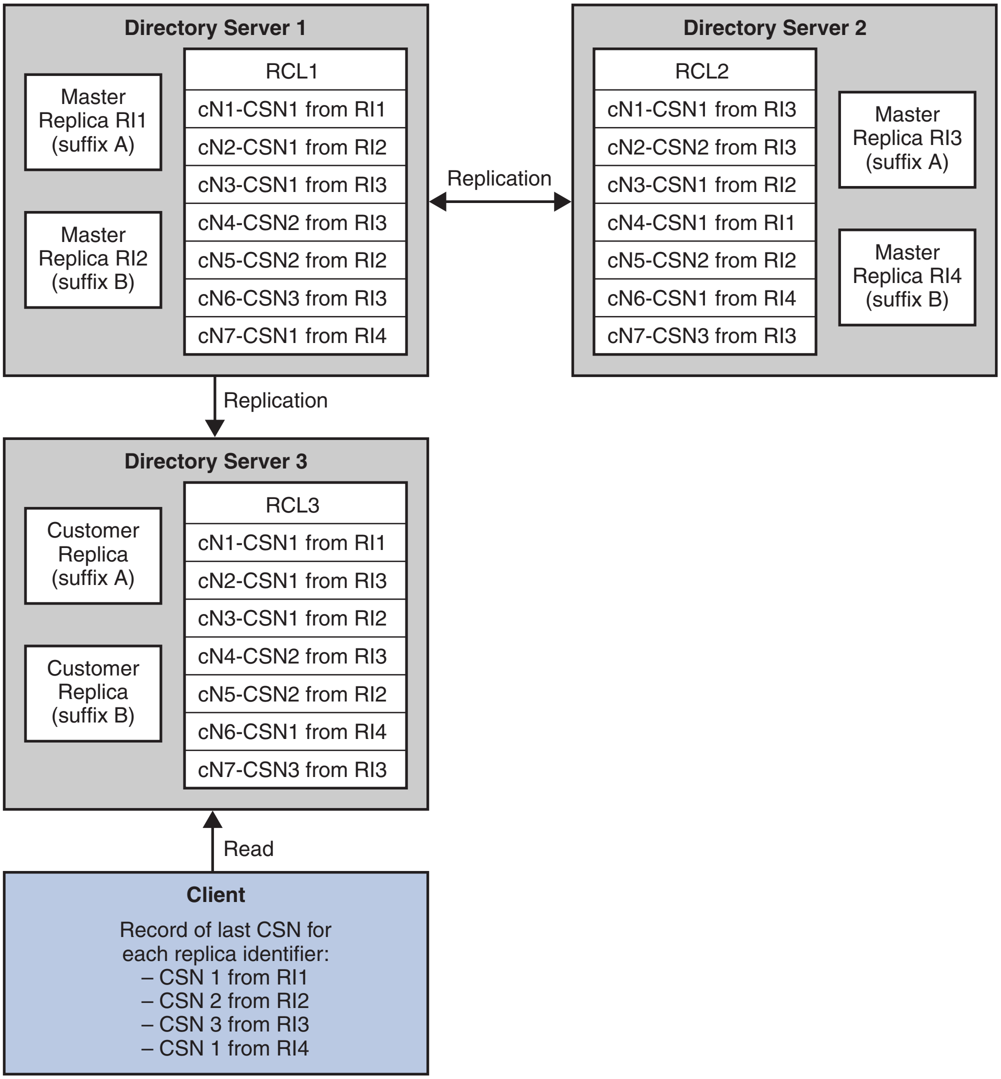
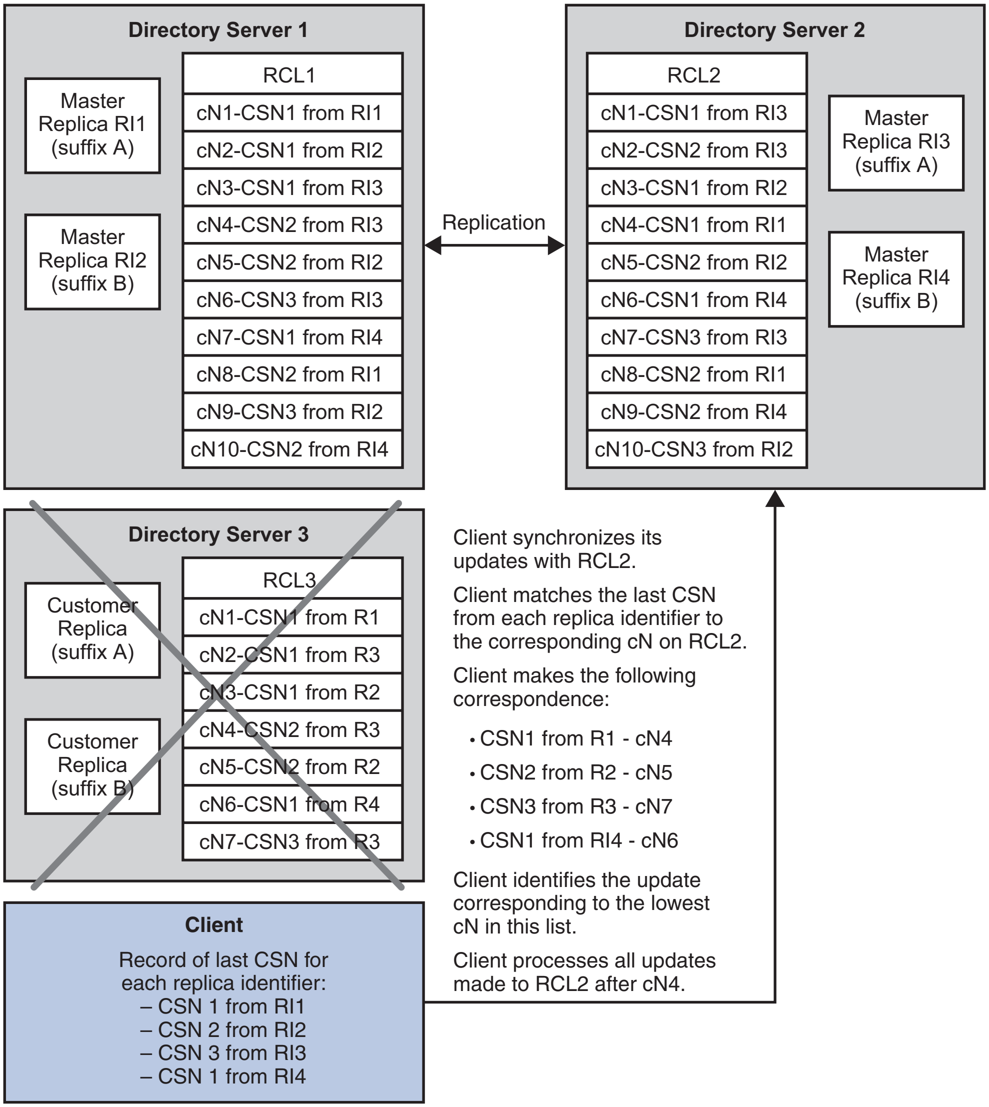


FIGURE 4–8 Simplified Topology for Replication of the Retro Change Log

All of the updates made to each master replica in the topology are logged to each retro change log in the topology.

The client application reads the retro change log of Directory Server 3 and stores the last CSN for each replica identifier. The last CSN for each replica identifier is given by the `replicationCSN` attribute.

The following figure shows the client redirecting its reads to Directory Server 2 after the failure of Directory Server 3.



Client synchronizes its updates with RCL2.

Client matches the last CSN from each replica identifier to the corresponding cN on RCL2.

Client makes the following correspondence:

- CSN1 from R1 - cN4
- CSN2 from R2 - cN5
- CSN3 from R3 - cN7
- CSN1 from RI4 - cN6

Client identifies the update corresponding to the lowest cN in this list.

Client processes all updates made to RCL2 after cN4.

FIGURE 4–9 Failover of the Retro Change Log

After failover, the client application must use the retro change log (RCL2) of Directory Server 2 to manage its updates. Because the order of the updates in RCL2 is not the same as the order in RCL3, the client must synchronize its updates with RCL2.

The client examines RCL2 to identify the cN that corresponds to its record of the last CSN for each replica identifier. In the example in "Failover of the Retro Change Log" on page 131, the client identifies the following correspondence between last CSN and cN:

- CSN 1 from R1 corresponds to cN4 on RCL2
- CSN 2 from R2 corresponds to cN5 on RCL2
- CSN 3 from R3 corresponds to cN7 on RCL2
- CSN 1 from R4 corresponds to cN6 on RCL2

The client identifies the update corresponding to the lowest cN in this list. In the example in "Failover of the Retro Change Log" on page 131, the lowest cN in the list is cN4. To ensure that the client processes all updates, it must process all updates logged to RCL2 after cN4. The client does not process updates logged to RCL2 before cN4 nor does it process the update corresponding to cN4.

Replication Conflicts and the Retro Change Log

When a replication conflict occurs, Directory Server performs operations to resolve the conflict. When the retro change log is running and the `changeIsReplFixupOp` attribute is set to `true`, the following information about the operations is logged in the `changeHasReplFixupOp` attribute:

- Target DN of the operation
- The type of update
- The change made

For more information about these attributes, see the *Sun Java System Directory Server Enterprise Edition 6.3 Man Page Reference*.

Restrictions on Using the Retro Change Log

Observe the following restrictions when you use the retro change log:

- A master replica running this version of Directory Server cannot be a supplier to a consumer replica running Directory Server 4.x.
- In a replicated topology, the retro change logs on replicated servers must be up-to-date with each other. This allows switchover of the retro change log. Using the example in "Failover of the Retro Change Log" on page 131, the last CSN for each replica ID on RCL3 must be present on RCL2.

CHAPTER 5

Directory Server Data Caching

For fast response time to client requests, Directory Server caches directory information in memory. If you must have top Directory Server performance, but cannot fit all directory data in available memory, you can tune cache settings to optimize performance.

This chapter covers what cache is, and also provides recommendations about tuning cache settings. This chapter includes the following sections:

- "Caches and How Directory Server Uses Them" on page 135
- "Tuning Cache Settings" on page 144

Caches and How Directory Server Uses Them

This section describes the types of cache whose settings you can tune. It also describes how Directory Server uses those types of cache. This section covers the following topics:

- "Types of Cache" on page 135
- "How Directory Server Performs Searches by Using Cache" on page 138
- "How Directory Server Performs Updates by Using the Cache" on page 140
- "How Directory Server Initializes a Suffix by Using the Cache" on page 142

Types of Cache

This section describes the types of cache used by Directory Server.

Figure 5–1 shows the caches for an instance of Directory Server with three suffixes, each with its own entry cache.

Directory Server also uses a file system cache. The file system cache is managed by the underlying operating system, and by I/O buffers in disk subsystems.

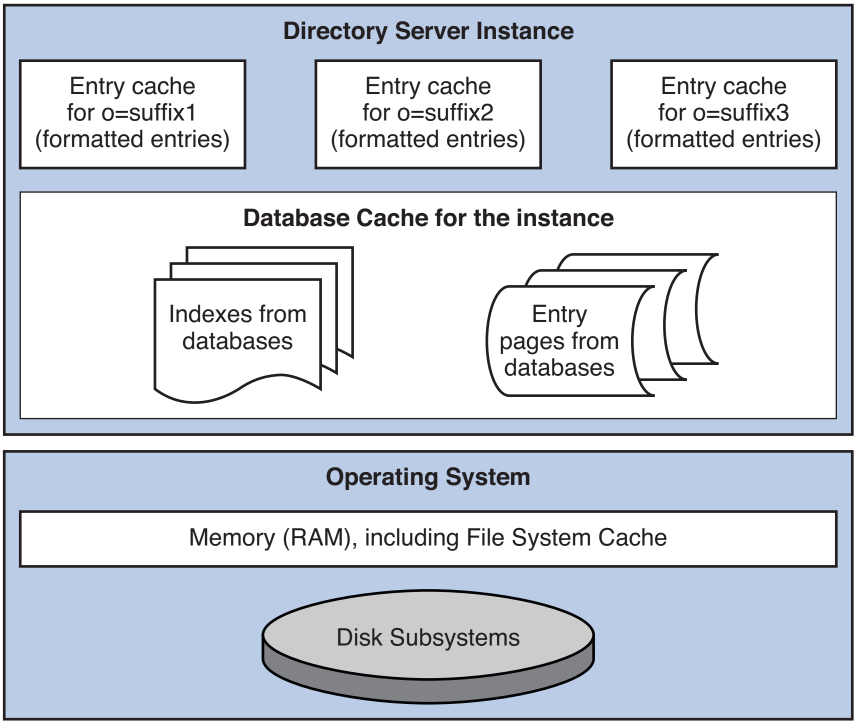

Directory Server Instance
Entry cache for o=suffix1 (formatted entries)
Entry cache for o=suffix2 (formatted entries)
Entry cache for o=suffix3 (formatted entries)
Database Cache for the instance
Indexes from databases
Entry pages from databases
Operating System
Memory (RAM), including File System Cache
Disk Subsystems

FIGURE 5–1 Entry and Database Caches in Context

Database Cache

Each instance of Directory Server has one database cache. The database cache holds pages from the database that contain indexes and entries. Each page is not an entry, but a slice of memory that contains a portion of the database.

Directory Server moves pages between the database files and the database cache to maintain the maximum database cache size you specify. The amount of memory used by Directory Server for the database cache can be larger than the specified size. This is because Directory Server requires additional memory to manage the database cache.

For very large database caches, it is important that the memory used by Directory Server does not exceed the size of available physical memory. If the available physical memory is exceeded, the system pages repeatedly and performance is degraded.

The memory can be monitored by empirical testing and by the use of tools such as `pmap`(1) on Solaris systems. The `ps`(1) utility can also be used with the `-p` *pid* and `-o` *format* options to view current memory used by a particular process such as Directory Server `ns-slapd`. For more information, refer to the operating system documentation.

For 32-bit servers, the database cache size must be limited so that the total Directory Server `ns-slapd` process size is less than the maximum process size allowed by the operating system. Usually, this limit is in the 2-3 GB range.

Entry Cache

The entry cache holds recently accessed entries that are formatted for delivery to client applications. The entry cache is allocated as required until it reaches a size larger than, but based on the maximum entry cache size you specify.

As entries stored in the entry cache are already formatted, Directory Server returns entries from an entry cache efficiently. Entries in the database must be formatted and stored in the entry cache before they are delivered to client applications.

The maximum size you specify indicates how much memory Directory Server requests from the underlying memory allocation library. Depending on how the memory allocation library handles requests for memory, the actual memory used may be much larger than the amount of memory available to Directory Server for the entry cache.

The memory used by the Directory Server process depends on the memory allocation library that is used, and depends on the entries cached. Entries with many small attribute values usually require more overhead than entries with few large attribute values.

For 32-bit servers, the entry cache size must be limited so that the total Directory Server `ns-slapd` process size is less than the maximum process size allowed by the operating system. In practice, this limit is generally in the 2-3 GB range.

Import Cache

The import cache is created and used when a suffix is initialized. If the deployment involves *offline* suffix initialization only, import cache and database cache are not used together. In this case, the import cache and database cache do not need to be added together when the cache size is aggregated. See "Total Aggregate Cache Size" on page 138. When the import cache size is changed, the change takes effect the next time the suffix is reset and initialized. The import cache is allocated for the initialization, then released after the initialization.

Directory Server handles import cache in the same way as it handles database cache. Sufficient physical memory must be available to prevent swapping. The benefits of having a larger import cache diminish for cache sizes larger than 2 GB.

File System Cache

The operating system allocates available memory not used by Directory Server caches and other applications to the file system cache. The file system cache holds data that was recently read from the disk, making it possible for subsequent requests to obtain data from cache rather than having to read it again from the disk. Because memory access is many times faster than disk access, leaving some physical memory available for the file system cache can boost performance.

For 32-bit servers, a file system cache can be used as a replacement for some of the database cache. Database cache is more efficient for Directory Server use than file system cache, but file system cache is not directly associated with the Directory Server `ns-slapd` process. Potentially, a larger total cache can be made available to Directory Server than would be available by using database cache alone.

64-bit servers do not have the same process size limit issue as 32-bit servers. Use database cache instead of file system cache with 64-bit servers.

Refer to the operating system documentation for information about file system cache.

Total Aggregate Cache Size

The sum of all caches used simultaneously must remain smaller than the total size of available physical memory, minus the memory intended for file system cache, minus the memory intended for other processes such as Directory Server itself.

For 32-bit servers, the total aggregate cache size must be limited so that the total Directory Server `ns-slapd` process size is less than the maximum process size allowed by the operating system. In practice, this limit is generally in the 2-3 GB range.

If suffixes are initialized while Directory Server is online, the sum of the database cache, the entry cache, and the import cache sizes should remain smaller than the total size of available physical memory.

TABLE 5–1 Import Operations and Cache Use

Cache Type	Offline Import	Online Import
Database	no	yes
Entry	yes	yes
Import	yes	yes

If all suffixes are initialized while Directory Server is offline, the import cache does not coexist with the database cache, so the same memory can be allocated to the import cache for offline suffix initialization and to the database cache for online use. If you opt to implement this special case, however, ensure that *no online bulk loads* are performed on a production server. The sum of the caches used simultaneously must remain smaller than the total size of available physical memory.

How Directory Server Performs Searches by Using Cache

In Figure 5–2, individual lines represent threads that access different levels of memory. Broken lines represent probable bottlenecks to minimize through effective tuning of Directory Server.

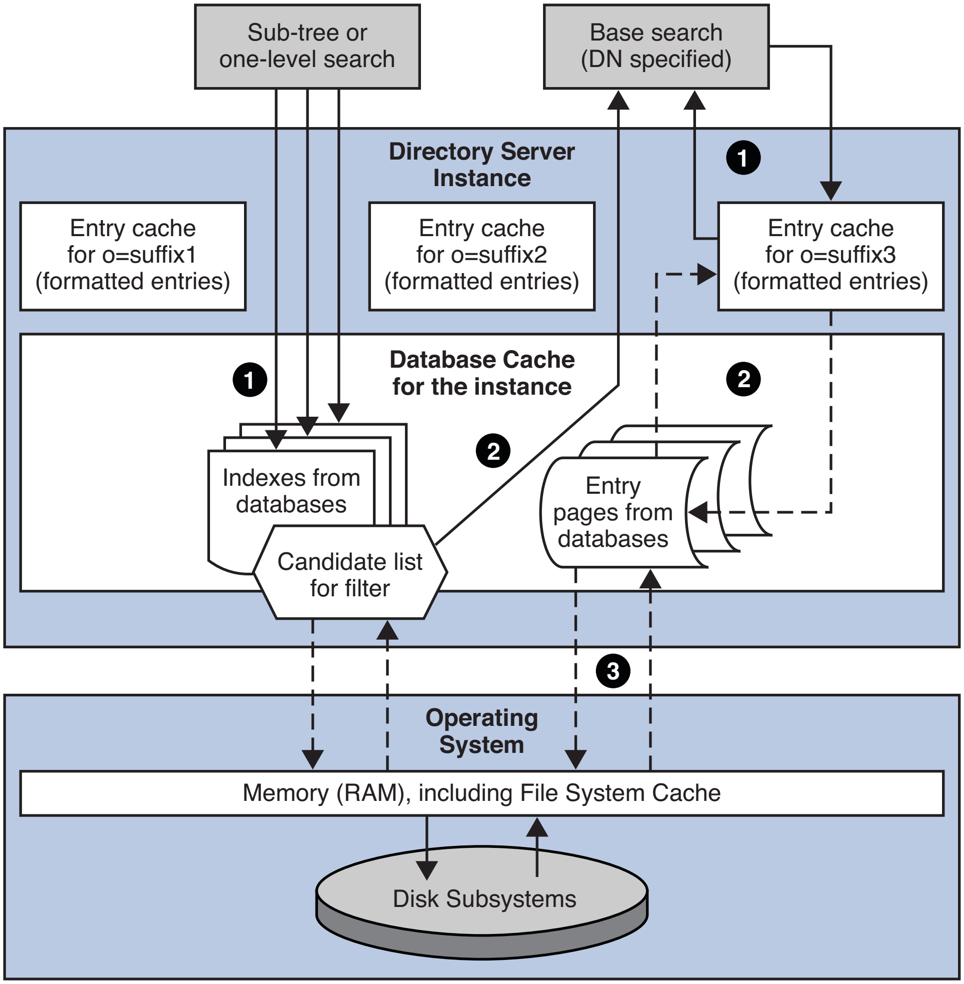

Sub-tree or one-level search
Base search (DN specified)
Directory Server Instance
1
Entry cache for o=suffix1 (formatted entries)
Entry cache for o=suffix2 (formatted entries)
Entry cache for o=suffix3 (formatted entries)
Database Cache for the instance
1
2
2
Indexes from databases
Candidate list for filter
Entry pages from databases
3
Operating System
Memory (RAM), including File System Cache
Disk Subsystems

FIGURE 5–2 How Directory Server Performs Searches

The following sections describe how Directory Server performs searches by using the cache. By processing subtree searches as described in the following sections, Directory Server returns results without loading the whole set of results into memory.

How Directory Server Performs Base Searches

Base searches specify a base DN and are the simplest type of searches for Directory Server to manage. Directory Server processes base searches in the following stages.

1. Directory Server attempts to retrieve the entry from the entry cache.

   If the entry is found in the entry cache, Directory Server checks whether the candidate entry matches the filter provided for the search.

   If the entry matches the filter provided for the search, Directory Server returns the formatted, cached entry to the client application.
2. Directory Server attempts to retrieve the entry from the database cache.

   If the entry is found in the database cache, Directory Server copies the entry to the entry cache for the suffix. Directory Server proceeds as if the entry had been found in the entry cache.
3. Directory Server attempts to retrieve the entry from the database itself.

   If the entry is found in the database, Directory Server copies the entry to the database cache . Directory Server proceeds as if the entry had been found in the database cache.

How Directory Server Performs Subtree and One-Level Searches

Searches on a subtree or a level of a tree involve additional processing to handle multiple entries. Directory Server processes subtree searches and one-level search in the following stages.

1. Directory Server attempts to define a set of candidate entries that match the filter from indexes in the database cache.

   If no appropriate index is present, the set of candidate entries must be found directly in the database itself.
2. For each candidate entry, Directory Server performs the following tasks.
   a. Performs a base search to retrieve the entry.
   b. Checks whether the entry matches the filter provided for the search.
   c. Returns the entry to the client application if the entry matches the filter.

How Directory Server Performs Updates by Using the Cache

In Figure 5–3, individual lines represent threads that access different levels of memory. Broken lines represent probable bottlenecks to minimize through effective tuning of Directory Server.

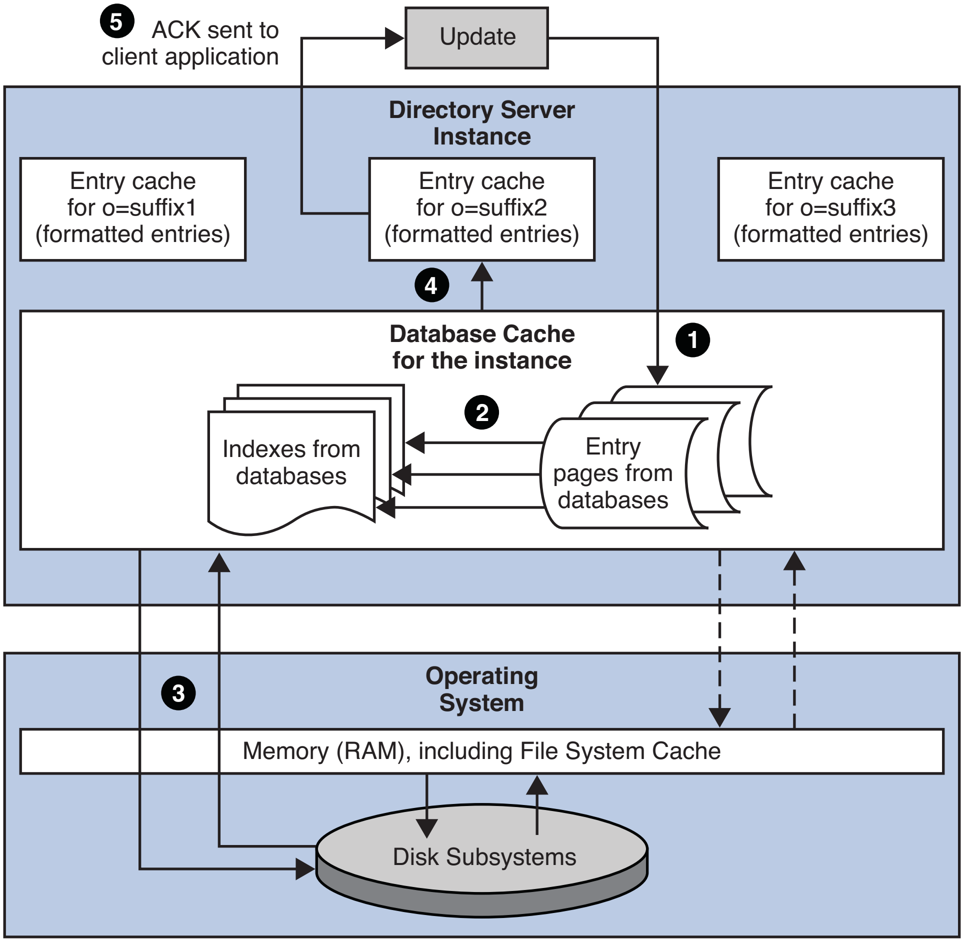

5
ACK sent to client application
Update
Directory Server Instance
Entry cache for o=suffix1 (formatted entries)
Entry cache for o=suffix2 (formatted entries)
Entry cache for o=suffix3 (formatted entries)
4
Database Cache for the instance
1
2
Indexes from databases
Entry pages from databases
3
Operating System
Memory (RAM), including File System Cache
Disk Subsystems

FIGURE 5–3 How Directory Server Performs Updates

The figure does not show the impact of the internal base search performed to get the entry for update.

Directory Server processes updates in the following stages.

1. Directory Server performs a base DN search to retrieve the entry, or to update or verify the entry in the case of an add operation that it does not already exist.
2. Directory Server updates the database cache and any indexes affected.

   If data affected by the change have not been loaded into the database cache, this step can result in disk activity while the relevant data are loaded into the cache.

3. Directory Server writes information about the changes to the transaction log and waits for the information to be flushed to disk, which happens periodically, at each checkpoint. Directory Server database files are thus updated during the checkpoint operation, not for each write.
4. Directory Server formats and copies the updated entry to the entry cache for the suffix.
5. Directory Server returns an acknowledgement of successful update to the client application.

How Directory Server Initializes a Suffix by Using the Cache

The following figure illustrates how Directory Server initializes a suffix by using the cache. Individual lines represent threads that access different levels of memory. Broken lines represent probable bottlenecks to minimize through effective tuning of Directory Server.

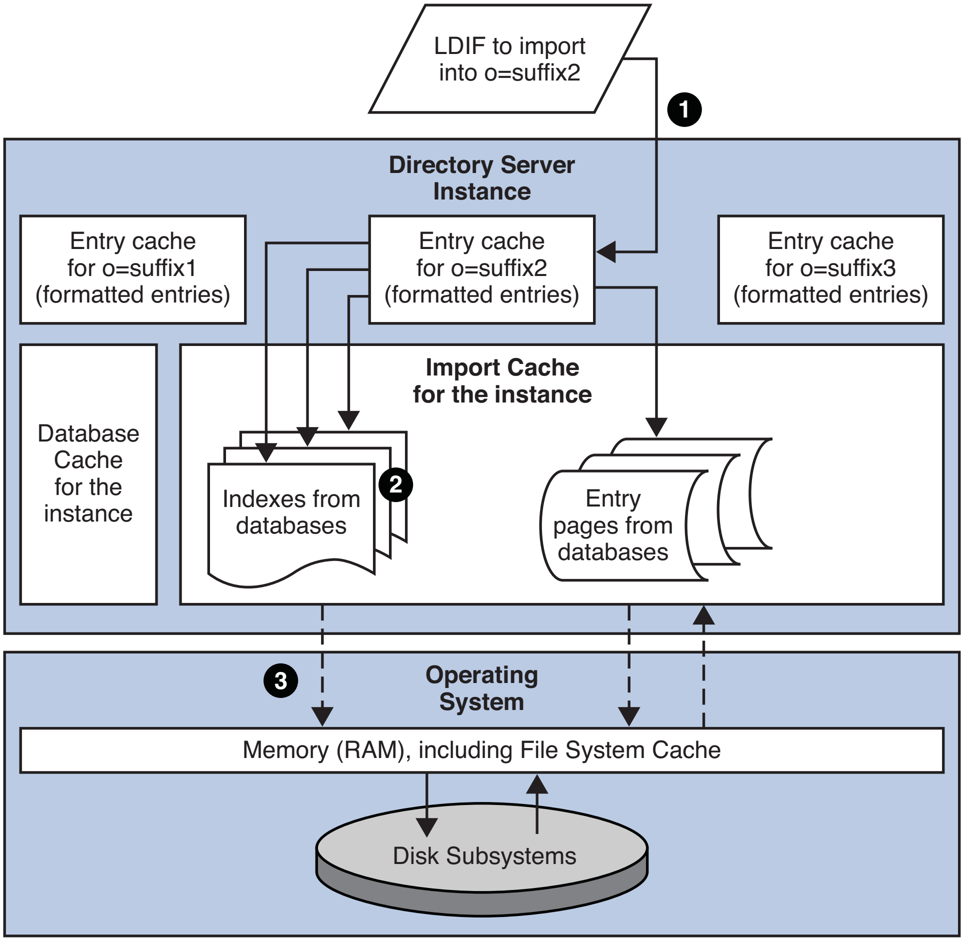

LDIF to import into o=suffix2
1
Directory Server Instance
Entry cache for o=suffix1 (formatted entries)
Entry cache for o=suffix2 (formatted entries)
Entry cache for o=suffix3 (formatted entries)
Database Cache for the instance
Import Cache for the instance
Indexes from databases
2
Entry pages from databases
3
Operating System
Memory (RAM), including File System Cache
Disk Subsystems

FIGURE 5–4 How Directory Server Initializes a Suffix

Directory Server initializes a suffix in the following stages:

1. Starts a thread to feed an entry cache, used as a buffer, from LDIF.
2. Starts a thread for each index affected and a thread to create entries in the import cache. These threads consume entries fed into the entry cache.
3. Reads from and writes to the database files when import cache runs out.

Directory Server can also write log messages during suffix initialization, but does not write to the transaction log.

Tools for suffix initialization delivered with Directory Server provide feedback on the cache hit rate and import throughput. If cache hit rate and import throughput drop together, it is possible that the import cache is too small.

Tuning Cache Settings

This section provides recommendations for setting database and entry cache sizes. It does not cover import cache sizes. The recommendations here pertain to maximizing either search rate or modify rate, not both at once.

This section covers the following topics:

- "Basic Tuning Recommendations" on page 144
- "Small, Medium, and Large Data Sets" on page 145
- "Optimum Search Performance (Searches Only)" on page 145
- "Optimum Modify Performance (Modifications Only)" on page 146

Basic Tuning Recommendations

Here you find the basic recommendations for maximizing search rates or maximizing modification rates achieved by Directory Server. Set cache sizes according to the following recommendations:

For Maximum Search Rate (Searches Only)

If the directory data do not fit into available physical memory, or only just fit with no extra room to spare, set cache sizes to their minimum values, `500k` for `db-cache-size`, `200k` for `entry-cache-size`, and allow the server to use as much of the operating system's file system cache as possible.

If the directory data fit into available physical memory with physical memory to spare, allocate memory to the entry cache until either the entry cache is full or, on a 32–bit system, the entry cache reaches maximum size. Then allocate memory to the database cache until it is full or reaches maximum size.

See "Configuring Memory" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide* for instructions on setting cache sizes.

For Maximum Modification Rate (Modifications Only)

If the directory data do not fit into available physical memory, or only just fit with no extra room to spare, set the entry cache sizes to the minimum value, `200k` for `entry-cache-size`, set the database cache to a value in the `100M` to `1G` range, and allow the server to use as much of the operating system's file system cache as possible. Keeping some database cache available ensures that modifications remain cached between each database checkpoint.

If the directory data fit into available physical memory with physical memory to spare, allocate memory to the entry cache until either the entry cache is full or, on a 32–bit system, the entry cache reaches maximum size. Then allocate memory to the database cache until it is full or reaches maximum size.

See "Configuring Memory" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide* for instructions on setting cache sizes.

Small, Medium, and Large Data Sets

A *working set* refers to the data actually pulled into memory so that the server can respond to client applications. The *data set* is then the entries in the directory that are being used due to client traffic. The data set may include every entry in the directory, or may be composed of some smaller number of entries, such as entries corresponding to people in a time zone where users are active.

We define three data set sizes, based on how much of the directory data set fits into available physical memory:

Small The data set fits entirely into physical memory with fully-loaded database and entry caches.

Medium The data set fits in physical memory, and extra physical memory can be dedicated to entry cache.

Large The data set is too small to fit completely in available physical memory.

The ideal case is of course the small data set. If your data set is small, set database cache size and entry cache size such that all entries fit in both the database cache and the entry cache.

The following sections provide recommendations for medium and large data sets where the server performs either all searches or all modify operations.

Optimum Search Performance (Searches Only)

Figure 5–5 shows search performance on a hypothetical system. As expected, Directory Server offers top search performance for a given system configuration when the whole data set fits into memory.

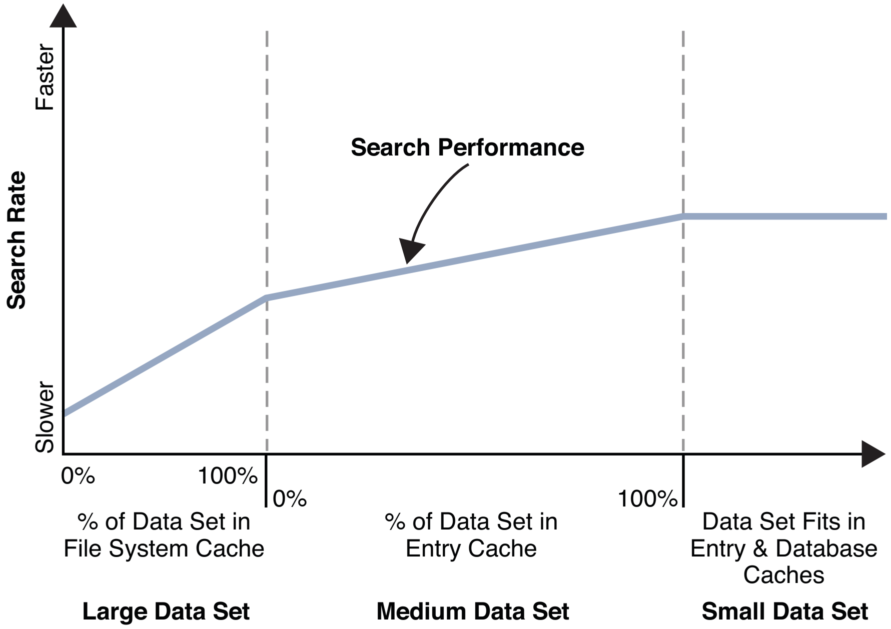


FIGURE 5–5 Search Performance

For large data sets better performance has been observed when database cache and entry cache are set to their minimum sizes and available memory is left to the operating system for use in allocating file system cache. As shown, performance improves when more of the data set fits into the file system cache.

For medium data sets better performance has been observed when the file system cache holds the whole data set, and extra physical memory available is devoted to entry cache. As shown, performance improves when more of the medium data set fits in entry cache.

Optimum Modify Performance (Modifications Only)

Figure 5–6 shows modify performance on a hypothetical system. As expected, Directory Server offers top modify performance for a given system configuration when the whole data set fits into memory.

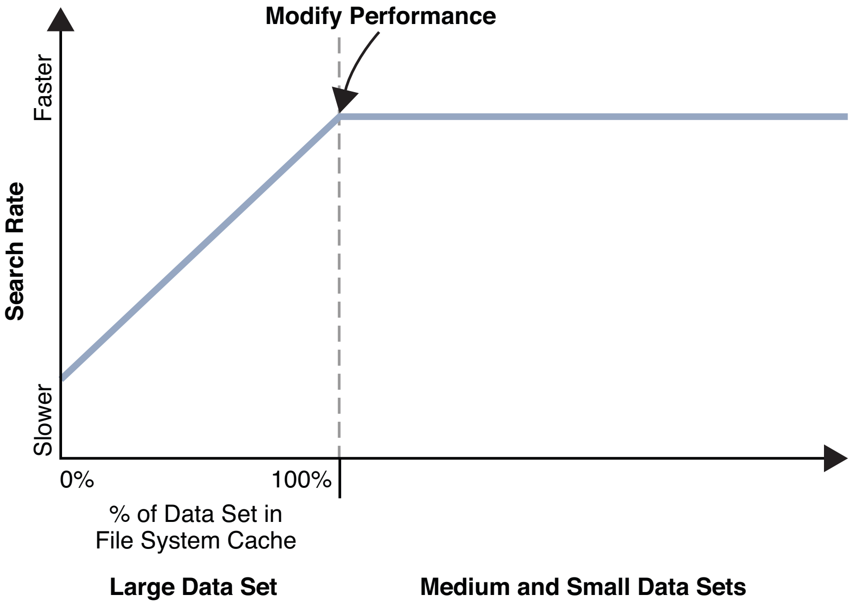


FIGURE 5–6 Modify Performance

For large data sets better performance has been observed when database cache and entry cache are set to their minimum sizes and available memory is left to the operating system for use in allocating file system cache. As shown, performance improves when more of the data set fits into the file system cache.

For medium data sets, modify performance reaches its maximum as all entries fit into file system cache. As suggested in "Basic Tuning Recommendations" on page 144, keeping some database cache available ensures the modifications to remain cached between each database checkpoint.

CHAPTER 6

Directory Server Indexing

Like a book index, Directory Server indexes speed up searches by associating search strings with the contents of a directory. For information about indexes used by Directory Server, see following sections:

Overview of Indexes

Directory Server uses indexes to speed up search operations by associating lookup information with Directory Server entries. During a search operation, Directory Server uses the index to find entries that match the search key . Without an index, Directory Server must check every entry in a suffix to find matches for the search key.

Indexes are stored in database files, and are created and managed independently for each suffix in a directory. Each index file contains all of the indexes defined in the suffix for a given attribute. For example, all indexes maintained for the cn attribute are stored in the *databaseName*_cn.db3 file. When an indexed entry is modified, Directory Server updates the index files.

Directory Server supports the following types of indexes:

- Default indexes to improve search performance or support searches performed by other applications. Default indexes are added when a suffix is created.
- System indexes to help Directory Server to function properly and efficiently.
- User indexes, added when a user creates an attribute or defines a new index.

Tuning Indexes for Performance

The use of indexes can enhance performance by reducing the time taken to perform a search. However, indexes also have an associated cost. When an entry is updated, the indexes referring to that entry must also be updated. The more an entry is indexed, the more resources are required to update the index; indexes take up disk space and memory space.

When you design indexes, ensure that you offset the benefit of faster searches against the associated costs of the index. Maintaining useful indexes is good practice; maintaining unused indexes for attributes on which clients rarely search is wasteful.

You can optimize performance of searches in the following ways:

- By preventing Directory Server from performing searches on non-indexed entries
- By limiting the length of an index list
- By ensuring that the size of a data base cache is appropriately tuned

To prevent Directory Server from performing searches on non-indexed entries, set the `require-index-enabled` suffix property for the suffix.

To limit the number of values per index key for a given search you can set an index list threshold. If the number of entries in the list for a search key exceeds the index list threshold, an unindexed search is performed. The threshold can be set for an entire server instance, for an entire suffix, and for an individual attribute type. You can also set individual thresholds for equality, presence, and substring indexes.

For a detailed procedure on how to change the index list threshold, see“Changing the Index List Threshold” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*. This procedure modifies the `all-ids-threshold` configuration property.

The global value of `all-ids-threshold` for the server instance should be about 5% of the total number of entries in the directory. For example, the default value of 4000 is generally right for instances of Directory Server that handle 80 000 entries or less. You should avoid setting the threshold above 50 000, even for very large deployments. However, you might set `all-ids-threshold` to a value other than the 5% guideline in the following situations:

- You expect the directory to grow considerably and wish to set the threshold higher than 5 percent of the total.
- You have consumers that support many searches and masters that support mostly writes, and you wish to set a different threshold for consumers and masters.
- You plan to initialize a large directory from an `LDIF` file and you wish to change the threshold just before initialization.
- Your directory has a deeply hierarchical directory information tree, and you are running one–level or subtree searches. In this case you could set the `all-ids-threshold` high for parent and ancestor indexes so that all entries below a given branch are indexed.

You should limit the number of unindexed searches that are performed. Use the `logconv` utility provided with the Directory Server Resource Kit to examine the access logs for evidence of frequent unindexed searches. For more information, see the `logconv(1)` man page.

System Indexes and Default Indexes

This section addresses the following topics:

- "System Indexes" on page 151
- "Default Indexes" on page 152

System Indexes

System indexes are required for Directory Server to function properly and efficiently. System indexes cannot be deleted or modified. Table 6–1 lists the system indexes created automatically in every suffix.

TABLE 6–1 System Indexes in Every Suffix

Attribute	Equality Index	Presence Index	Description
`aci`		X	Allows the directory server to quickly obtain the access control information maintained in the directory
`ancestorid`	X		Enhances directory performance during subtree searches
`entrydn`	X		Speeds up entry retrieval based on DN searches
`id2entry`	X		Contains the actual database of directory entries. All other database files can be recreated from this one
`nsUniqueId`	X		Used to search for specific entries
`nscpEntryDN`	X		Used internally in Directory Server for replication
`nsds5ReplConflict`	X	X	Helps to find replication conflicts
`numsubordinates`		X	Used by Directory Service Control Center to enhance display performance on the Directory tab
`objectClass`	X		Accelerate subtree searches
`parentID`	X		Enhances directory performance during one-level searches

Default Indexes

When you create a new suffix in your directory, the server configures a set of default indexes in the corresponding database directory. The default indexes can be modified depending on your indexing needs, although you should ensure that no server plug-ins or other servers in your enterprise depend on an indexed attribute before you eliminate index.

Table 6–2 lists the default indexes that are configured in Directory Server.

TABLE 6–2 Default Indexes in Every New Suffix

Attribute	Equality Index	Presence Index	Substring Index	Description
cn	X	X	X	Improves the performance of the most common types of directory searches.
givenName	X	X	X	Improves the performance of the most common types of directory searches.
mail	X	X	X	Improves the performance of the most common types of directory searches.
mailAlternateAddress	X			Used by Messaging Server.
mailHost	X			Used by Messaging Server.
member	X			Improves server performance. This index is also used by the referential integrity plug-in.
nsCalXItemId	X	X	X	Used by Calendar Server.
nsLIProfileName	X			Used by roaming feature of Messaging Server.
nsRoleDN	X			Improves the performance of role-based operations.
nswcalCALID	X			Used by Calendar Server.
owner	X			Improves server performance. This index is also used by the referential integrity plug-in.
pipstatus	X			Used by other servers.
pipuid		X		Used by other servers.
seeAlso	X			Improves server performance. This index is used by the referential integrity plug-in.
sn	X	X	X	Improves the performance of the most common types of user directory searches.

TABLE 6–2 Default Indexes in Every New Suffix *(Continued)*

Attribute	Equality Index	Presence Index	Substring Index	Description
telephoneNumber	X	X	X	Improves the performance of the most common types of user directory searches.
uid	X			Improves server performance.
uniquemember	X			Improves server performance. This index is also used by the referential integrity plug-in.

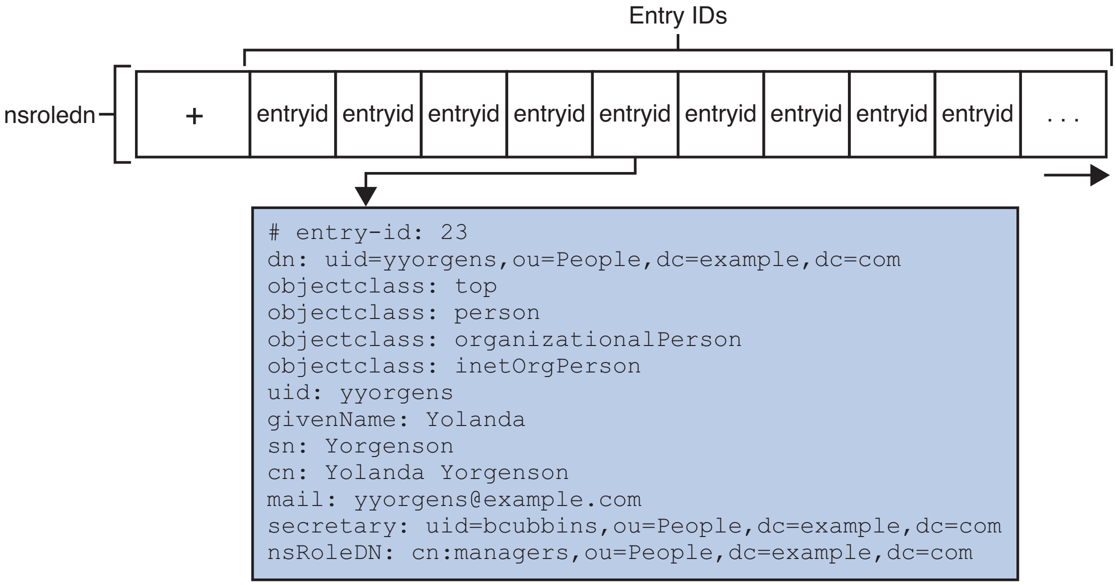
Types of Index

With the exception of the approximate index, the indexes in this section are used by Directory Server to speed up basic matching rules. This section covers the following index types:

- "Presence Index" on page 153
- "Equality Index" on page 154
- "Substring Index" on page 156
- "Browsing Index" on page 157
- "Approximate Index" on page 158
- "International Index" on page 159

Presence Index

The presence index includes all entries in the database that have a value for a specified attribute, irrespective of that value. The following figure shows a presence index for the nsRoleDN attribute. For information about this attribute, see nsRoleDN(5dsat).



```
# entry-id: 23
dn: uid=yyorgens,ou=People,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
uid: yyorgens
givenName: Yolanda
sn: Yorgenson
cn: Yolanda Yorgenson
mail: yyorgens@example.com
secretary: uid=bcubbins,ou=People,dc=example,dc=com
nsRoleDN: cn:managers,ou=People,dc=example,dc=com
```

FIGURE 6–1 Presence Index

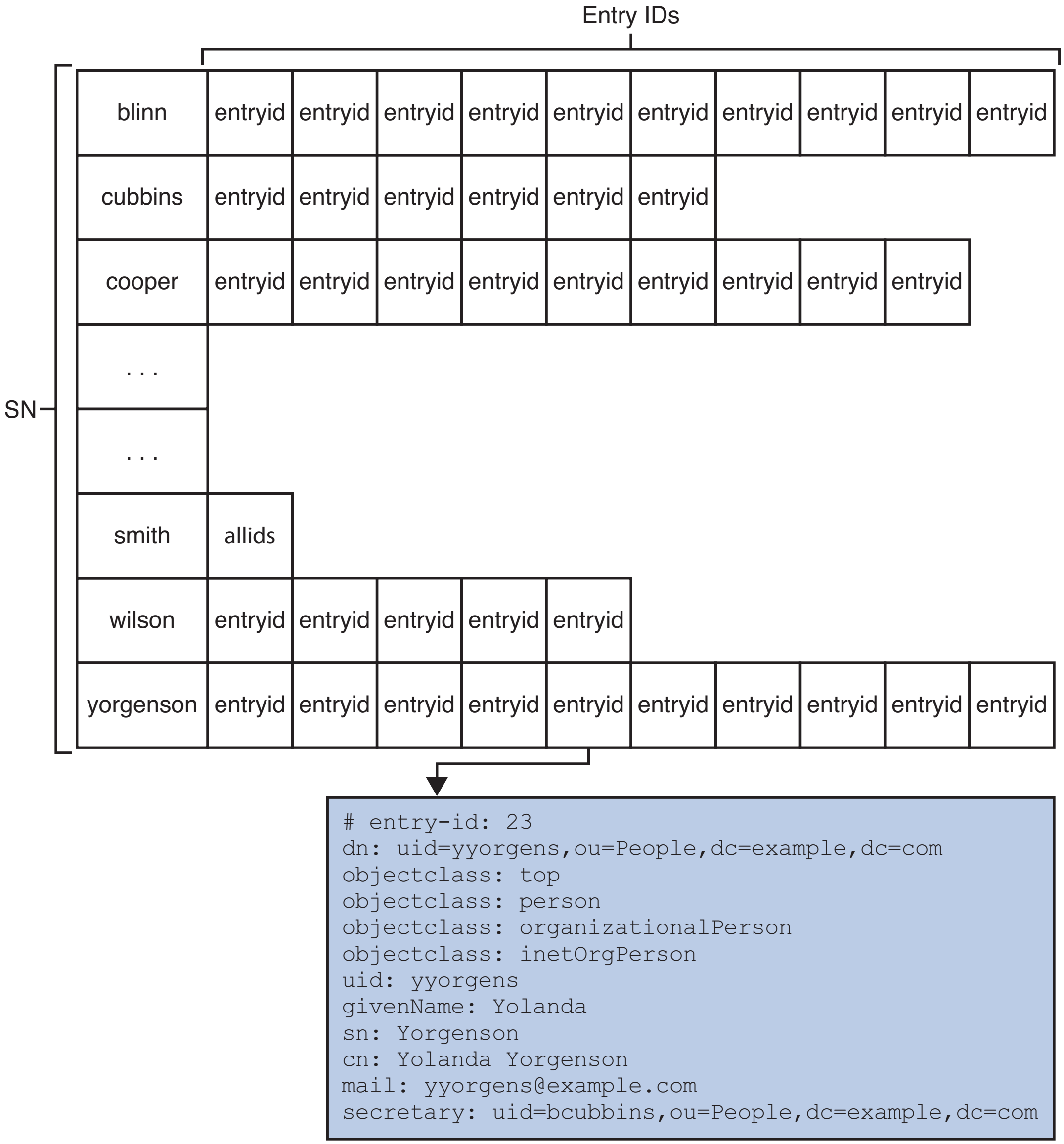
Directory Server uses the value of the `entryid` attribute to store a reference to the entry. Directory Server retrieves the entry by using the *instance-path*/db/*dbinstance*/*dbinstance*_id2entry.db3 index file, where *dbinstance* depends on the database identifier.

When Directory Server receives a request to remove an attribute value indexed for presence, it must remove the entry from the presence index for that attribute before acknowledging the update to the client application.

The cost of presence indexes is generally low, although the list of entries maintained for a presence index may be long. When the index list length is small, presence indexes are useful for attributes in a relatively small percentage of directory entries.

Equality Index

The equality index includes all entries in the database that have a specified value for a given attribute. This index requires a value to be specified in the search filter. The following figure shows an equality index for the `sn`, surname, attribute. The index maintains a list of values for the `sn` attribute. For information about this attribute, see `sn`(5dsat).



blinn	entryid	entryid	entryid	entryid	entryid	entryid	entryid	entryid	entryid	entryid
cubbins	entryid	entryid	entryid	entryid	entryid	entryid				
cooper	entryid	entryid	entryid	entryid	entryid	entryid	entryid	entryid	entryid	
. . .										
. . .										
smith	allids									
wilson	entryid	entryid	entryid	entryid	entryid					
yorgenson	entryid	entryid	entryid	entryid	entryid	entryid	entryid	entryid	entryid	entryid

```
# entry-id: 23
dn: uid=yyorgens,ou=People,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
uid: yyorgens
givenName: Yolanda
sn: Yorgenson
cn: Yolanda Yorgenson
mail: yyorgens@example.com
secretary: uid=bcubbins,ou=People,dc=example,dc=com
```

FIGURE 6–2 Equality Index

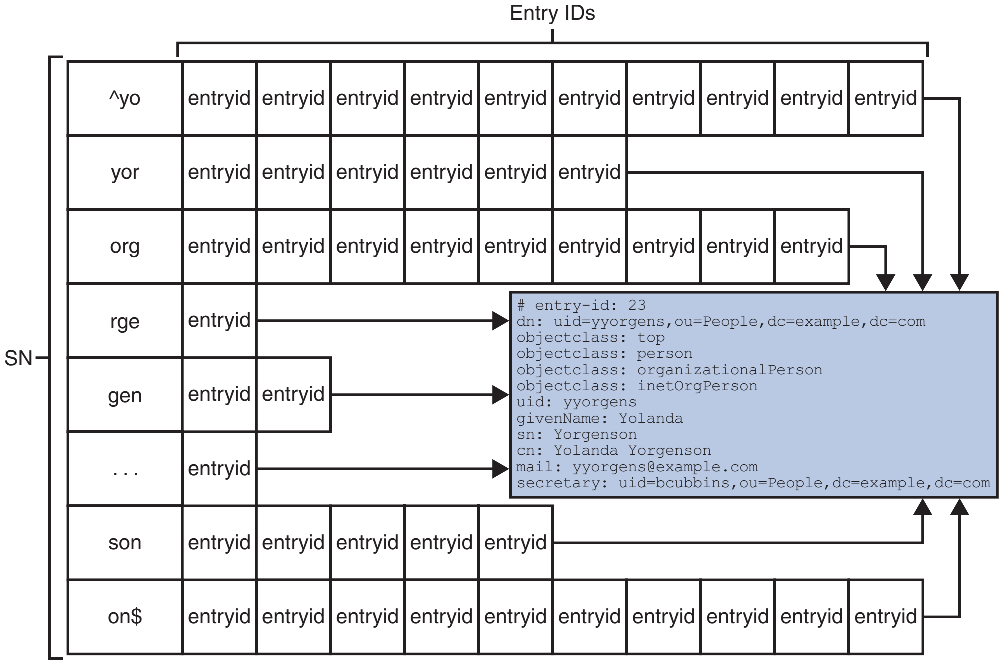
When Directory Server receives a request to update an entry indexed for equality, it must do the following tasks before performing the update and acknowledging the update to the client:

- Determine whether the entry must be removed from the index
- Determine whether a list must be added to or removed from the index

The cost of equality indexes is generally lower than for substring indexes, but equality indexes require more space than presence indexes. Some client applications such as messaging servers might rely on equality indexes for search performance. Avoid using equality indexes for large binary attributes such as photos and hashed passwords.

Substring Index

Substring indexes are used for searches on three-character groups, for example, `sn=*abc*`. The three-character groups are stored in the index. Substring indexes cannot be applied to binary attributes such as photos. The following figure shows a substring index for the `SN` attribute.



```
# entry-id: 23
dn: uid=yyorgens,ou=People,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
uid: yyorgens
givenName: Yolanda
sn: Yorgenson
cn: Yolanda Yorgenson
mail: yyorgens@example.com
secretary: uid=bcubbins,ou=People,dc=example,dc=com
```

FIGURE 6–3 Substring Index for the `SN` Attribute

The Directory Server search algorithm includes optimizations for the following searches, however, these searches are more likely to reach the index list threshold:

- Searches on two-character substrings with this format `sn=*ab*`
- Searches on one-character group with this format `sn=a*`. Searches cannot be performed on one-character groups with this format `sn=*a` and `sn=*a*`

Directory Server builds an index of substrings according to its own built-in rules. Substring indexes cannot be configured by the system administrator.

When Directory Server receives a request to update an entry that has an attribute indexed for substrings, it must do the following tasks before performing the update and acknowledging the update to the client:

- Determine whether the entry must be removed from the index
- Determine whether and how modifications to the entry affect the index
- Determine whether the entry IDs or lists of entry IDs must be added to or removed from the index

Maintaining substring indexes is relatively costly; the cost is a function of the length of the string indexed. To minimize cost, avoid unnecessary substring indexes, especially for attributes that have potentially long string values such as a description.

Browsing Index

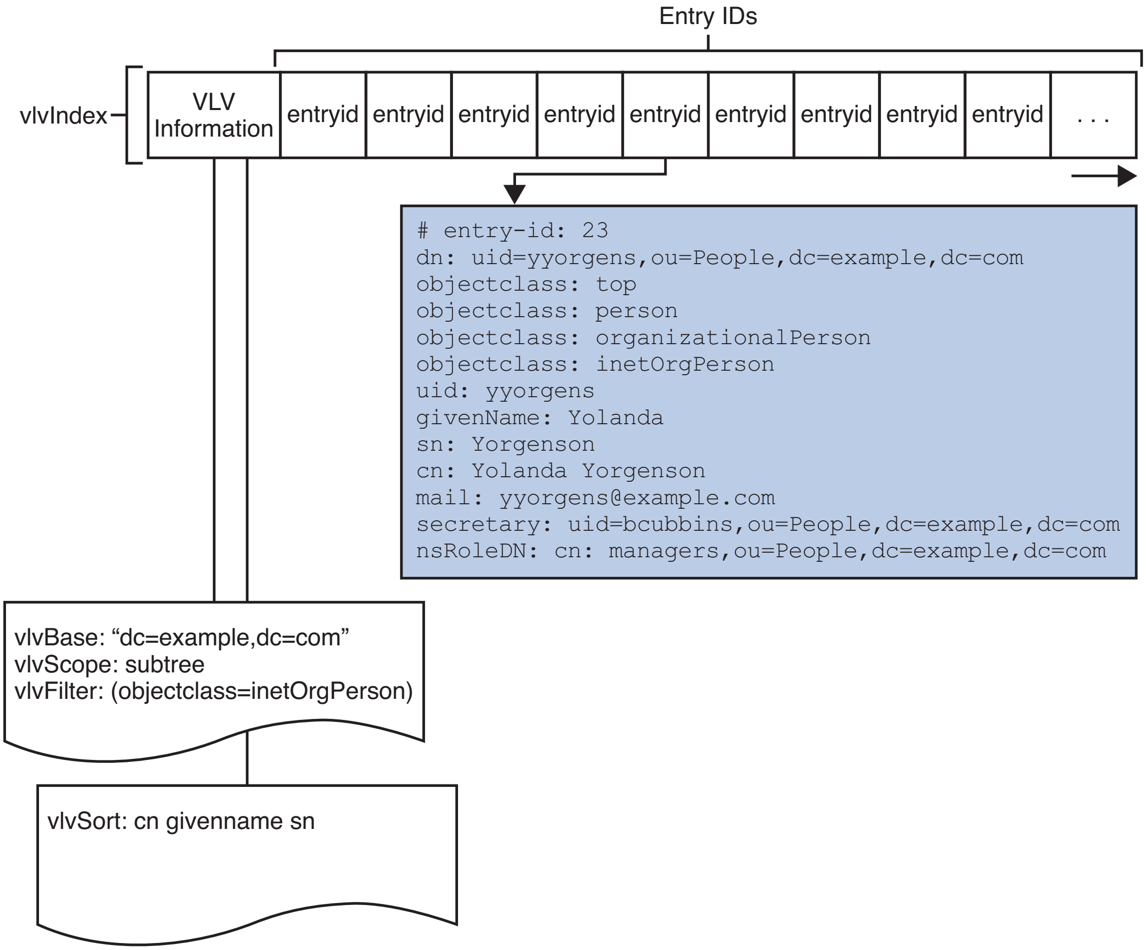
Browsing indexes are also called *virtual list view indexes*. Browsing indexes are used for search operations that request server-side sorting or virtual list view, VLV, results. By using browsing indexes, you can improve the performance of searches that request server-side sorting of a large number of results. Depending on your directory configuration, the server may refuse to perform searches that request sorting when no browsing index is defined. This prevents large sorting operations from overloading server resources.

Browsing indexes are configured with the following parameters in the `vlvSearch`(5dsoc) object class, `vlvBase`(5dsat)`vlvScope`(5dsat), `vlvScope`(5dsat), and `vlvFilter`(5dsat). Browsing index are sorted by the following parameter in the `vlvIndex`(5dsoc) object class, `vlvSort`(5dsat).

Browsing indexes are configured in two steps.

1. The base of the search, the scope of the search, and a filter for the search are configured by the `vlvBase`, `vlvScope`, and `vlvFilter` attributes in the `vlvSearch` object class.
2. The name of the attributes that sort the index are configured by the `vlvSort` attribute in the `vlvIndex` object class.

The following figure shows a browsing index.



```
# entry-id: 23
dn: uid=yyorgens,ou=People,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
uid: yyorgens
givenName: Yolanda
sn: Yorgenson
cn: Yolanda Yorgenson
mail: yyorgens@example.com
secretary: uid=bcubbins,ou=People,dc=example,dc=com
nsRoleDN: cn: managers,ou=People,dc=example,dc=com
```

vlvBase: “dc=example,dc=com”
vlvScope: subtree
vlvFilter: (objectclass=inetOrgPerson)

vlvSort: cn givenname sn

FIGURE 6–4 Representation of a Browsing Index

When Directory Server receives a request to update an entry with a `vlvFilter` value, it must do the following tasks before performing the update and acknowledging the update to the client:

- Determine whether the entry must be removed from the index
- Determine the correct position of the entry in the list

Approximate Index

Approximate indexes work with the English language only to provide efficient “sounds-like” searches. For example, the approximate index is useful for searching partial names or misspelled names. Directory Server uses a variation of the metaphone phonetic algorithm to perform searches on an approximate index. Because the algorithm is based loosely on syllables, it is not effective for attributes that contain numbers, such as telephone numbers.

International Index

International indexes are also called matching rule indexes. International indexes associate language-specific matching rules with attributes. This index type enables attributes to be sorted and searched for in accordance with the language rules. International indexes use matching rules for particular locales to maintain indexes.

Standard support for international and other types of indexing can be extended by using a custom matching rule server plug-in.

CHAPTER 7

Directory Server Logging

For information about the types of logs used in Directory Server and for a description of the server logs, see the following sections:

- "Introduction to Logs" on page 161
- "Retro Changelog" on page 162
- "Transaction Log" on page 162
- "Access, Error, and Audit Logs" on page 163

Introduction to Logs

The following table summarizes the different logs used by the Directory Server.

TABLE 7–1 Logs Used by Directory Server

Log	Type	Description
Retro change log	Database	Maintaining application compatibility with earlier versions of Directory Server.
Transaction log	Database	Ensuring data integrity by committing each update operation to the transaction log on disk before the result code for the update operation is returned to the client application. When Directory Server accepts an update operation, it writes a log message about the operation to the transaction log. If the system crashes, Directory Server uses the transaction log to recover the database.
Access log	Flat file	Evaluating directory use patterns, verifying configuration settings, diagnosing access problems. For information about access logs, see "Access Logs" on page 163.
Error log	Flat file	Debugging directory deployments. For information about error logs, see "Error Logs" on page 164.

TABLE 7–1 Logs Used by Directory Server *(Continued)*

Log	Type	Description
Audit log	Flat file	Providing audit trails for security and data integrity. For information about audit logs, see "Audit Logs" on page 164.

Retro Changelog

The following server properties configure the retro change log.

`retro-cl-deleted-entry-attr`
Attributes to record when an entry is deleted

`retro-cl-enabled`
Whether the retro changelog is enabled

`retro-cl-ignored-attr`
Attributes not to record when updates occur

`retro-cl-max-age`
Maximum age of records maintained

`retro-cl-max-entry-count`
Maximum total records maintained

`retro-cl-path`
File system directory where the log is housed

`retro-cl-suffix-dn`
Suffixes for which the log is maintained

See `server`(5dsconf) for details.

Transaction Log

The following server properties configure the transaction log.

`db-checkpoint-interval`
How often Directory Server checkpoints the transaction log, ensures the entire database system is synchronized to disk, and cleans up transaction logs

`db-durable-transaction-enabled`
Whether update operations are committed to the transaction log on disk before result codes are sent to clients

`db-log-buf-size`
The buffer size for log information stored in memory until the buffer fills or the transaction commit forces the buffer to be written to disk

`db-log-path`
The path of the transaction log

`db-batched-transaction-count`
How many updates are accumulated before being committed to the directory database

See `server`(5dsconf) for details.

Access, Error, and Audit Logs

Access logs, error logs and audit logs are flat files that contain information about operations. For information about how to view and configure logs, see Chapter 15, "Directory Server Logging," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

By default, the logs are stored in the directory *instance-path*`/logs/`.

Log files can be rotated on demand, or can be scheduled to be rotated on a specific day-of-the week and time of day, or when the log file exceeds a specified minimum size.

Old log files are stored in the same path with the same name and an extension that contains the date that the file was created, in the format *filename*`.`*YYYYMMDD*`-`*hhmmss*. The server also maintains a file with the same name and the `.`*rotationinfo* extension to record the creation dates of all log files.

For information about access logs, error logs and audit logs, see the following sections:

- "Access Logs" on page 163
- "Error Logs" on page 164
- "Audit Logs" on page 164
- "Content of Access, Error, and Audit Logs" on page 164
- "Connection Codes in Log Files" on page 169
- "Result Codes in Log Files" on page 170

Access Logs

Access logs contain information about connections between an LDAP client and a directory server. A connection is a sequence of requests from the same client, and can contain the following components:

- Connection index and the IP address of the client
- Bind record
- Bind result record
- Sequence of operation request/result pairs, or individual records in the case of connection, closed, and abandon records

- Unbind record
- Closed record

Error Logs

Error logs contain a unique identifier of the error, warning or information message, and a human readable message. Errors are defined according to the following severity.

Error — The error is severe. Immediate action should be taken to avoid the loss or corruption of directory data.

Warning — The error is important. Action should be taken at some stage to prevent a severe error occurring in the future.

Info — An informative message, usually describing server activity. No action is necessary.

Audit Logs

Audit logs contain records of all modifications to configuration or suffix entries. The modifications are written in LDIF format.

Audit logging is not enabled by default. To enable audit logging, use the procedure "To Enable the Audit Log" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Content of Access, Error, and Audit Logs

The remainder of this chapter describes each of the parts of the log files.

Time Stamp

Each line of an access log file begins with a timestamp of this format:`[20/Dec/2006:11:39:51 -0700]`. The time stamp, `-0700` indicates the time difference in relation to GMT.

The format of the time stamp can vary according to your platform. The connection, closed, and abandon records appear individually. All other records appear in pairs, consisting of a request for service record followed by a result record. The record pairs usually, but not exclusively, appear on adjacent lines.

Connection Number

The connection number is represented by `conn=`*value*. Every external request is listed with an incremental connection number.

When `conn=Internal` the operation is an internal operation. To log internal access operations, specify an access logging level of `acc-internal` in the `dsconf` configuration attribute.

File Descriptor

The file descriptor is represented by `fd=`*value*.

Every connection from an external LDAP client to a directory server requires a file descriptor from the operating system. The file descriptor is taken from a pool of available file descriptors.

Slot Number

The slot number has the same meaning as file descriptor. Slot number is a legacy section of the access log and can be ignored.

Operation Number

The operation number is represented by `op=`*value*.

For a connection, all operation request and result pairs are given incremental operation numbers beginning with `op=0`. The operation number identifies the operation being performed.

When `op=-1`, the LDAP request for the connection was not issued by an external LDAP client, but was initiated internally.

Method Type

The method type is represented by `method=`*value*.

The method type indicates which bind method was used by the client. The method type can have one of the following values.

`0`	No authentication
`128`	Simple bind with user password
`sasl`	SASL bind using external authentication mechanism

LDAP Version

The LDAP version can be `LDAPv2` or `LDAPv3`. The LDAP version gives the LDAP version number that the LDAP client used to communicate with the LDAP server.

Error Number

The error number is represented by `err=`*number*.

The error number provides the LDAP result code returned from the LDAP operation. The LDAP error number `0` means that the operation was successful. For a list of LDAP result codes refer to "Result Codes in Log Files" on page 170.

Tag Number

The tag number is represented by `tag=`*value*.

The tags are used internally for message decoding and are not intended for use outside. The following tags are used most often.

`tag=97`	A client bind operation
`tag=100`	The entry for which you were searching
`tag=101`	The result from a search operation
`tag=103`	The result from a modify operation
`tag=105`	The result from an add operation
`tag=107`	The result from a delete operation
`tag=109`	The result from a modify DN operation
`tag=111`	The result from a compare operation
`tag=115`	A search reference when the entry you perform your search on holds a referral to the entry you require. Search references are expressed in terms of a referral.
`tag=120`	A result from an extended operation

Number of Entries

The number of entries is represented by `nentries=`*value*.

The number of entries indicates the number of entries that matched an LDAP search request.

Elapsed Time

The elapsed time is represented by `etime=`*value*.

Elapsed time indicates the time that it took to perform the LDAP operation. An `etime` value of `0` means that the operation took milliseconds to perform.

To log the time in microseconds, specify an access logging level of `acc-timing` in the `dsconf` configuration attribute.

LDAP Request Type

The LDAP request type indicates the type of LDAP request made by the client. The following types of LDAP requests can be made:

`SRCH`	Search
`MOD`	Modify
`DEL`	Delete
`ADD`	Add
`MODDN`	Modify DN
`EXT`	Extended operation
`ABANDON`	Abandon operation

LDAP Response Type

The LDAP response type indicates the LDAP response being returned by the server. The following LDAP responses can be returned:

`RESULT`	Result
`ENTRY`	Entry
`REFERRAL`	Referral or search reference

Unindexed Search Indicator

The unindexed search indicator is represented by `notes=U`.

In an unindexed search, the database is searched instead of the index file. Unindexed searches occur for the following reasons:

- The all IDs threshold was reached in the index file used for the search
- An index file does not exist
- The index file is not configured in the way required by the search

An unindexed search indicator is often accompanied by a large `etime` value because unindexed searches are usually more time consuming than indexed searches.

Extended Operation OID

An extended operation OID is represented by `EXT oid="`*OID number*`"`. See `extended-operations`(5dsconf) for a list of supported extended operations.

Change Sequence Number in Log Files

The replication change sequence number is represented in log files by `csn=`*value*.

The presence of a change sequence number indicates that replication is enabled for this naming context.

Abandon Message

The abandon message is represented by `ABANDON`.

The presence of the abandon message indicates that an operation has been aborted. If the message ID succeeds in locating the operation that has been aborted, the log message reads as follows:

```
conn=12 op=2 ABANDON targetop=1 msgid=2 nentries=0 etime=0
```

However, if the message ID does not succeed in locating the operation, or if the operation had already finished prior to the `ABANDON` request being sent, then the log message reads as follows:

```
conn=12 op=2 ABANDON targetop=NOTFOUND msgid=2
```

The abandon message uses the following parameters:

`nentries` Gives the number of entries sent before the operation was aborted

`etime` Gives the number of seconds that elapsed before the operation was aborted

`targetop` Identifies the operation to be aborted. If the value is `NOTFOUND`, the operation to be aborted was either an unknown operation or already complete

Message ID

The message ID is represented by `msgId=`*value*.

The message ID is the LDAP operation identifier generated by the client. The message ID can have a different value to the operation number, but identifies the same operation. The message ID in an `ABANDON` operation specifies which client operation is being abandoned.

The operation number starts counting at 0. However, in many client implementations the message ID number starts counting at 1. This explains why the message ID is frequently equal to the operation number plus 1.

SASL Multi-Stage Bind Logging

Directory Server logs each stage in the multi stage bind process and, where appropriate, the progress statement `SASL bind in progress` is included.

The DN used for access control decisions is logged in the BIND result line and not in the bind request line.

```
conn=14 op=1 RESULT err=0 tag=97 nentries=0 etime=0 dn="uid=myname,dc=example,dc=com"
```

For SASL binds, the DN value displayed in the BIND request line is not used by the server and is, therefore, not relevant. However, for SASL binds, the authenticated DN must be used for audit purposes. Therefore, the authenticated DN must be clearly logged. Having the authenticated DN logged in the BIND result line avoids any confusion as to which DN is which.

Options Description

The options description, `options=persistent`, indicates that a persistent search is being performed. Persistent searches can be used as a form of monitoring and can be configured to return changes to given configurations. The access log distinguishes between persistent and regular searches.

Connection Codes in Log Files

A connection code is included in the `closing` message of a log file. The connection code provides additional information about why the connection was closed. The following table describes the common connection codes.

TABLE 7–2 Summary of Connection Codes

Connection Code	Description
A1	The client has closed the connection without performing an UNBIND.
B1	This connection code can have one of the following causes: ■ The client has closed the connection without performing an UNBIND. ■ The BER element was corrupt. If BER elements, which encapsulate data being sent over the wire, are corrupt when they are received, a B1 connection code is logged to the access log. BER elements can be corrupted by physical layer network problems or bad LDAP client operations, such as an LDAP client aborting before receiving all request results.
B2	The BER element is longer than the `nsslapd-maxbersize` attribute value.
B3	A corrupt BER tag was encountered.
B4	The server failed to flush data response back to client. This code can occur when the client closes the connection to the server, before the server finished sending data to the client.
P1	The client connection was closed by a custom plug-in. None of the plug-ins provided by Directory Server close a connection.

TABLE 7–2 Summary of Connection Codes *(Continued)*

Connection Code	Description
P2	A closed connection or corrupt connection has been detected.
T1	The server closed the client connection because it was idle for longer than the `idle-timeout` server property.
T2	The server closed the client connection because it was stalled for longer than the `nsslapd-ioblocktimeout` attribute value. This code can occur for the following reasons: ■ There is a network problem. ■ The server sends a lot of data to the client but the client does not read the data. As a result, the server's transmit buffer becomes full.
U1	The server closed the client connection because client sent an UNBIND request.

Result Codes in Log Files

The following tables summarizes the LDAP result codes generated by an LDAP server and an LDAP client.

TABLE 7–3 Summary of Result Codes for LDAP Servers

Result Code	Description
0	Success
1	Operations error
2	Protocol error
3	Time limit exceeded
4	Size limit exceeded
5	Compare false
6	Compare true
7	Authentication method not supported
8	Strong authentication required
9	Partial results and referral received
10	Referral received
11	Administrative limit exceeded
12	Unavailable critical extension

TABLE 7–3 Summary of Result Codes for LDAP Servers *(Continued)*

Result Code	Description
13	Confidentiality required
14	SASL bind in progress
16	No such attribute
17	Undefined attribute type
18	Inappropriate matching
19	Constraint violation
20	Type or value exists
21	Invalid syntax
32	No such object
33	Alias problem
34	Invalid DN syntax
35	Object is a leaf
36	Alias de-referencing problem
48	Inappropriate authentication
49	Invalid credentials
50	Insufficient access
51	Server is busy
52	Server is unavailable
53	Server is unwilling to perform
54	Loop detected
64	Naming violation
65	Object class violation
66	Operation not permitted on a non-leaf entry
67	Operation not permitted on a RDN
68	Entry already exists
69	Cannot modify object class
70	Results too large
71	Affects multiple servers

TABLE 7–3 Summary of Result Codes for LDAP Servers *(Continued)*

Result Code	Description
76	Virtual list view error

TABLE 7–4 Summary of Result Codes for LDAP Clients

Result Code	Description
80	Unknown error
81	Cannot contact LDAP server
82	Local error
83	Encoding error
84	Decoding error
85	Timed out
86	Unknown authentication method
87	Bad search filter
88	User cancelled operation
89	Bad parameter to an LDAP routine
90	Out of memory
91	Cannot connect to the LDAP server
92	Not supported by this version of LDAP
93	Requested LDAP control not found
94	No results returned
95	Additional results to return
96	Client detected loop
97	Referral hop limit exceeded

CHAPTER 8

Directory Server Groups and Roles

This chapter describes how groups and roles are used by Directory Server to associate entries with each other. This chapter covers the following topics:

- "Directory Server Groups" on page 173
- "Directory Server Roles" on page 174

Directory Server Groups

A group is an entry that identifies the other entries that are in the group. The group mechanism makes it easy to retrieve a list of entries that are members of a given group. Directory Server supports static and dynamic groups.

Static Groups

Static groups specify the DN of each member of the group. Static groups use one of the following object class and attribute pairs:

- The `groupOfNames` object class, with a multivalued `member` attribute
- The `groupOfUniqueNames` object class, with a multivalued `uniqueMember` attribute

The `member` attribute and `uniqueMember` attribute contain the DN for every entry that is a member of the group. The `uniqueMember` attribute value for the DN is optionally followed by a hash, `#`, and a unique identifier label to guarantee uniqueness.

Dynamic Groups

Dynamic groups specify one or more URL search filters. All entries that match the URL search filters are members of the group. Membership of a dynamic group is defined each time the filters are evaluated. Dynamic groups use one of the following object class and attribute pairs:

- The `groupOfURLs` object class, with the `memberURL` attribute
- The `groupOfUniqueNames` object class, with the `uniqueMember` attribute

The group members are listed either by one or more filters represented as LDAP URL values of the `memberURL` attribute or by one or more DNs as values of the `uniqueMember` attribute.

Nested Groups

Static and dynamic groups can be nested by specifying the DN of another group as a value for the `member` attribute or `uniqueMember` attribute. The depth to which nested groups are supported by ACIs is controlled by the `nsslapd-groupevalnestlevel` configuration parameter.

Directory Server Roles

Roles are similar to groups but work in the opposite way — where a group entry lists the DN of the member entries, the DN of a role entry is listed on each member entry. The role mechanism makes it is easy to retrieve a list of roles that are assigned to an entry.

Each role has *members*, or entries that possess the role. The role mechanism is managed by the `nsRoleDN` attribute and the `nsRole` attribute. The `nsRoleDN` attribute is used to add an entry to a role. The `nsRole` attribute is a read-only attribute, maintained by the directory server, that lists the roles to which an entry belongs. The `nsRole` attribute can be read or searched by clients to enumerate all roles to which an entry belongs. If you do not want to expose role membership, define access controls to read-protect the `nsRole` attribute.

By default, the scope of a role is limited to the subtree where it is defined. The scope of a role can be extended to other subtrees on the same server instance.

Managed Roles

Managed roles are functionally very similar to static groups. Managed roles explicitly assign a role to each member entry by adding the `nsRoleDN` attribute to the entry. The value of this attribute is the DN of the role definition entry.

The role definition entry only defines the scope of the role in the directory. Members of the role are entries that lie within the scope of the role definition, and that identify the role definition entry with their `nsRoleDN` attributes.

Filtered Roles

Filtered roles are equivalent to dynamic groups. Entries are assigned a role if they match a specified search filter. The value of the search filter is defined by the `nsRoleFilter` attribute. When the server returns an entry in the scope of a filtered role, that entry contains the generated `nsRole` attribute that identifies the role.

Nested Roles

Nested roles are equivalent to nested groups. Nested roles enable you to create roles that contain other roles and to extend the scope of existing roles. A nested role can itself contain another nested role. Up to 30 levels of nesting are supported

A nested role lists the definition entries of other roles and combines all the members of their roles. If an entry is a member of a role that is listed in a nested role, then the entry is also a member of the nested role.

Limitations on Using Roles

When you use roles to support your directory service, be aware of the following limitations.

Roles and chaining

> If your directory tree is distributed over several servers by using the chaining feature, entries that define roles must be located on the same server as the entries that possess those roles. If one server, A, receives entries from another server, B, through chaining, those entries will contain the roles defined on B, but will not be assigned any of the roles defined on A.

Filtered Roles cannot use CoS generated attributes

> The filter string of a filtered role cannot be based on the values of a CoS virtual attribute. However, the specifier attribute in a CoS definition may reference the `nsRole` attribute generated by a role definition. For information about CoS, see Chapter 9, “Directory Server Class of Service.”

Extending the scope of roles

> You can extend the scope of roles to different subtrees but they must be on the same server instance. You cannot extend the scope of roles to other servers.

Searches on the `nsRole` attribute

> The `nsRole` attribute can be used in any search filter with any of the comparison operators. When you search on `nsRole` attribute, consider the following points:
>
> - Searches on the `nsRole` attribute can take a long time because all roles must be evaluated before the entries can be filtered.
> - Directory Server is optimized for equality searches on membership in managed roles. For example, this search will be nearly as fast as a search on real attributes.

```
(&(nsRole=cn=managersRole,ou=People,dc=example,dc=com)
 (objectclass=person)
```

- The nsRoleDN attribute is indexed by default in all suffixes. Optimizations for searching the membership of managed roles are lost if indexing is disabled for the nsRoleDN attribute.
- Searches for entries that contain a filtered role involve an internal search with the role filter. This internal operation will be fastest if all attributes that appear in the role filter are indexed in all suffixes in the scope of the role.

CHAPTER 9

Directory Server Class of Service

The Class of Service (CoS) mechanism allows attributes to be shared between entries. CoS values are calculated dynamically when they are requested. For information about CoS, see the following sections:

About CoS

Imagine a directory containing thousands of entries that all have the same value for the `facsimileTelephoneNumber` attribute. Traditionally, to change the fax number, you would update each entry individually, a time consuming job for administrators. Using CoS, the fax number is stored in a single place, and the `facsimileTelephoneNumber` attribute is automatically generated on every entry as it is returned.

To client applications, a CoS attribute is generated in the same ways as any other attribute. However, directory administrators now have only a single fax value to manage. Also, because there are fewer values stored in the directory, the database uses less disk space. The CoS mechanism also allows entries to override a generated value or to generate multiple values for the same attribute.

Note – Because CoS virtual attributes are not indexed, referencing them in an LDAP search filter may have an impact on performance.

Generated CoS attributes can be multivalued. Specifiers can designate several template entries, or there can be several CoS definitions for the same attribute. Alternatively, you can specify template priorities so that only one value is generated from all templates.

Roles and classic CoS can be used together to provide role-based attributes. These attributes appear on an entry because it possesses a particular role with an associated CoS template. You could use a role-based attribute to set the server look through limit on a role-by-role basis, for example.

CoS functionality can be used recursively; you can generate attributes through CoS that depend on other attributes generated through CoS. Complex CoS schemes can simplify client access to information and ease administration of repeated attributes, but they also increase management complexity and degrade server performance. Avoid overly complex CoS schemes; many indirect CoS schemes can be redefined as classic or pointer CoS, for example.

You should also avoid changing CoS definitions more often than necessary. Modifications to CoS definitions do not take effect immediately, because the server caches CoS information. Although caching accelerates read access to generated attributes, when changes to CoS information occur, the server must reconstruct the cache. This task can take some time, usually in the order of seconds. During cache reconstruction, read operations may still access the old cached information, rather than the newly modified information, which means that if you change CoS definitions too frequently, you are likely to be accessing outdated data.

CoS Definition Entries and CoS Template Entries

The CoS mechanism relies on two types of entries, the CoS definition entry and the CoS template entry. This section describes the CoS definition entry and the CoS template entry.

CoS Definition Entry

The CoS definition entry identifies the type of CoS and the names of the CoS attributes that will be generated. Like the role definition entry, the CoS definition entry inherits from the `LDAPsubentry` object class. Multiple definitions may exist for the same CoS attribute, which, as a result, may be multivalued.

The CoS definition entry is an instance of the `cosSuperDefinition` object class. The CoS definition entry also inherits from one of the following object classes to specify the type of CoS:

- `cosPointerDefinition`
- `cosIndirectDefinition`
- `cosClassicDefinition`

The CoS definition entry contains the attributes specific to each type of CoS for naming the virtual CoS attribute, the template DN, and the specifier attribute in target entries. By default, the CoS mechanism will not override the value of an existing attribute with the same name as the CoS attribute. However, the syntax of the CoS definition entry allows you to control this behavior.

When schema checking is turned on, the CoS attribute will be generated on all target entries that allow that attribute. When schema checking is turned off, the CoS attribute will be generated on all target entries.

The location of the definition entry determines the scope of the CoS, which is the entire subtree below the parent of the CoS definition entry. All entries in the branch of the definition entry's parent are called *target entries* for the CoS definition.

The following figure shows a CoS definition entry at the root of the `ou=people` subtree. The scope of the CoS is only the two subtrees beneath the root. The CoS does not extend above this root, or to other subtrees in the DIT.

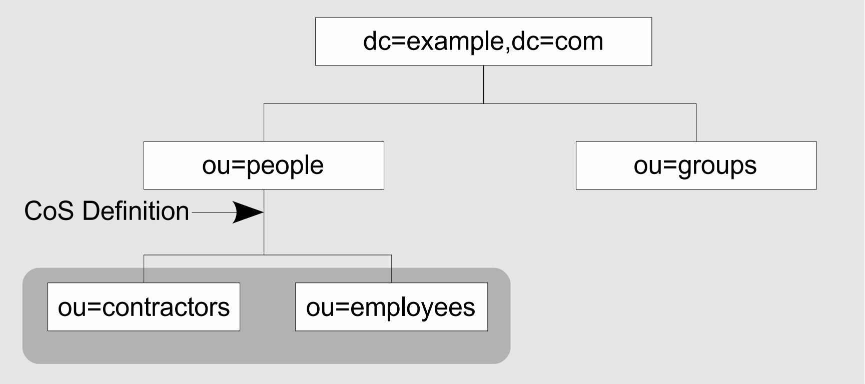


FIGURE 9–1 CoS Scope

CoS Template Entry

The CoS template entry contains the value that is generated for the CoS attribute. All entries within the scope of the CoS use the values defined here. There may be several templates, each with a different value, in which case the generated attribute may be multivalued. The CoS mechanism selects one of these values based on the contents of both the definition entry and the target entry.

The CoS template entry is an instance of the `cosTemplate` object class. The CoS template entry contains the value or values of the attributes generated by the CoS mechanism. The template entries for a given CoS are stored in the directory tree at the same level as the CoS definition.

When possible, definition and template entries should be located in the same place, for easier management. You should also name them in a way that suggests the functionality they provide. For example, a definition entry DN such as `"cn=classicCosGenEmployeeType,ou=People,dc=example,dc=com"` is more descriptive than `"cn=ClassicCos1,ou=People,dc=example,dc=com"`. For more information about the object classes and attributes associated with each type of CoS, see "Class of Service" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Pointer CoS, Indirect CoS, and Classic CoS

The following types of CoS differ in how the template, and therefore the generated value, is selected:

- "Pointer CoS" on page 180
- "Indirect CoS" on page 181
- "Classic CoS" on page 182

Pointer CoS

Pointer CoS is the simplest type of CoS. The pointer CoS definition entry provides the DN of a specific template entry of the cosTemplate object class. All target entries have the same CoS attribute value, as defined by this template.

The following figure shows a pointer CoS that defines a common postal code for all of the entries stored under dc=example,dc=com. The CoS definition entry, CoS template entry and target entry are indicated.

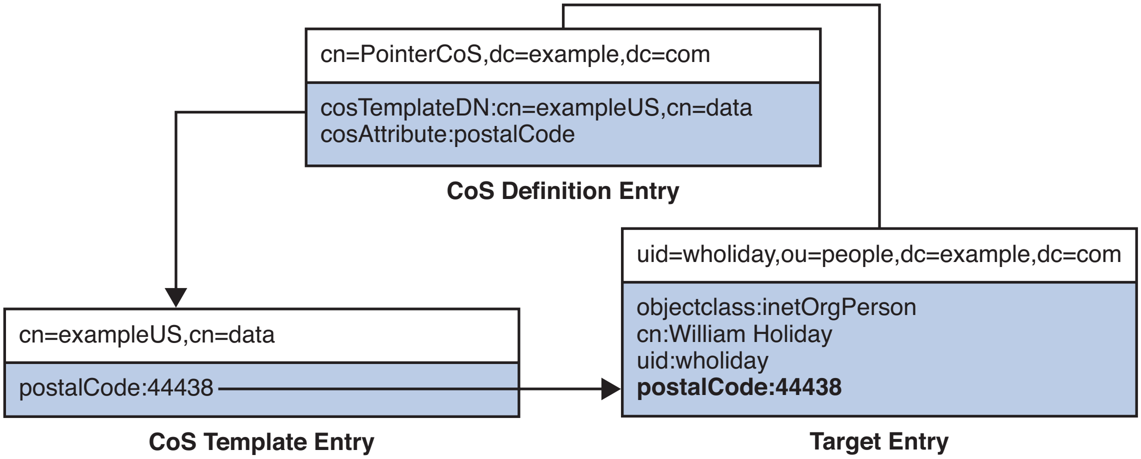


FIGURE 9–2 Example of a Pointer CoS Definition and Template

The template entry is identified by its DN, cn=exampleUS,cn=data, in the CoS definition entry. Each time the postalCode attribute is queried on entries under dc=example,dc=com, Directory Server returns the value available in the template entry cn=exampleUS,cn=data. Therefore, the postal code will appear with the entry uid=wholiday,ou=people,dc=example,dc=com, but it is not stored there.

In a scenario where several shared attributes are generated by CoS for thousands or millions of entries, instead of existing as real attributes in each entry, the storage space savings and performance gains provided by CoS are considerable.

Indirect CoS

Indirect CoS allows any entry in the directory to be a template and provide the CoS value. The indirect CoS definition entry identifies an attribute, called the indirect specifier, whose value in a target entry determines the template used for that entry. The indirect specifier attribute in the target entry must contain a DN. With indirect CoS, each target entry may use a different template and thus have a different value for the CoS attribute.

For example, an indirect CoS that generates the `departmentNumber` attribute may use an employee's manager as the specifier. When retrieving a target entry, the CoS mechanism will use the DN value of the `manager` attribute as the template. It will then generate the `departmentNumber` attribute for the employee using the same value as the manager's department number.

Note – Because templates may be arbitrary entries anywhere in the directory tree, implementing access control for indirect CoS can become extremely complex. In deployments where performance is critical, you should also avoid overusing indirect CoS due to its resource intensive nature.

In many cases, results that are similar to those made possible by indirect CoS can be achieved by limiting the location of the target entries with classic CoS or using the less flexible pointer CoS mechanism.

The following figure shows an indirect CoS that uses the `manager` attribute of the target entry to identify the template entry. In this way, the CoS mechanism can generate the `departmentNumber` attribute of all employees to be the same as their manager's, ensuring that it is always up to date.

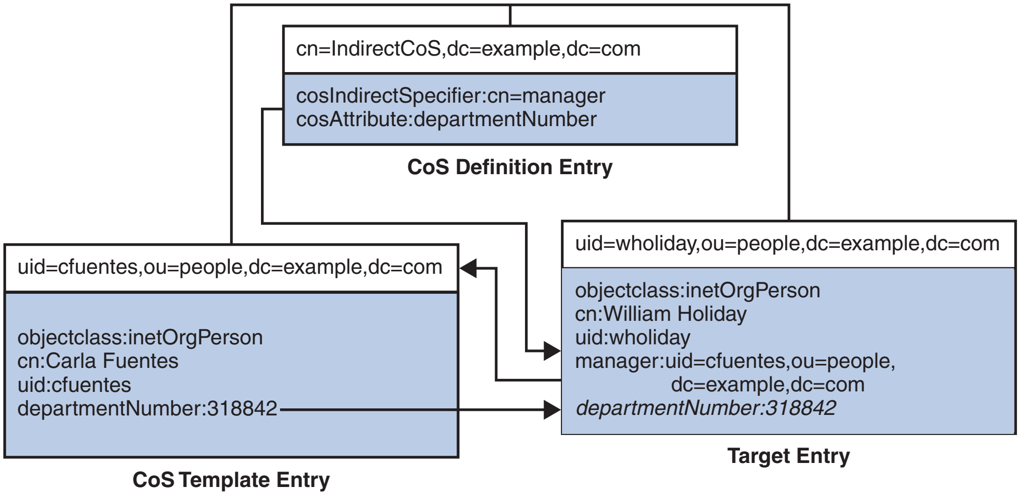

cn=IndirectCoS,dc=example,dc=com
cosIndirectSpecifier:cn=manager
cosAttribute:departmentNumber
CoS Definition Entry
uid=cfuentes,ou=people,dc=example,dc=com
objectclass:inetOrgPerson
cn:Carla Fuentes
uid:cfuentes
departmentNumber:318842
CoS Template Entry
uid=wholiday,ou=people,dc=example,dc=com
objectclass:inetOrgPerson
cn:William Holiday
uid:wholiday
manager:uid=cfuentes,ou=people,
dc=example,dc=com
departmentNumber:318842
Target Entry

FIGURE 9–3 Example of an Indirect CoS Definition and Template

The indirect CoS definition entry names the specifier attribute, which in this example, is the `manager` attribute. William Holiday's entry is one of the target entries of this CoS, and his `manager` attribute contains the DN of `uid=cfuentes,ou=people,dc=example,dc=com`. Therefore, Carla Fuentes's entry is the template, which in turn provides the `departmentNumber` attribute value of `318842`.

Classic CoS

Classic CoS combines the pointer and indirect CoS behavior. The classic CoS definition entry identifies the base DN of the template and a specifier attribute. The value of the specifier attribute in the target entries is then used to construct the DN of the template entry as follows:

`cn=`*specifierValue*`,` *baseDN*

The template containing the CoS values is determined by the combination of the RDN (relative distinguished name) value of the specifier attribute in the target entry and the template's base DN.

Classic CoS templates are entries of the `cosTemplate` object class to avoid the performance issue associated with arbitrary indirect CoS templates.

The classic CoS mechanism determines the DN of the template from the base DN given in the definition entry and the specifier attribute in the target entry. The value of the specifier attribute is taken as the `cn` value in the template DN. Template DNs for classic CoS must therefore have the following structure:

`cn=`*specifierValue*`,`*baseDN*

The following figure shows a classic CoS definition that generates a value for the postal code attribute.

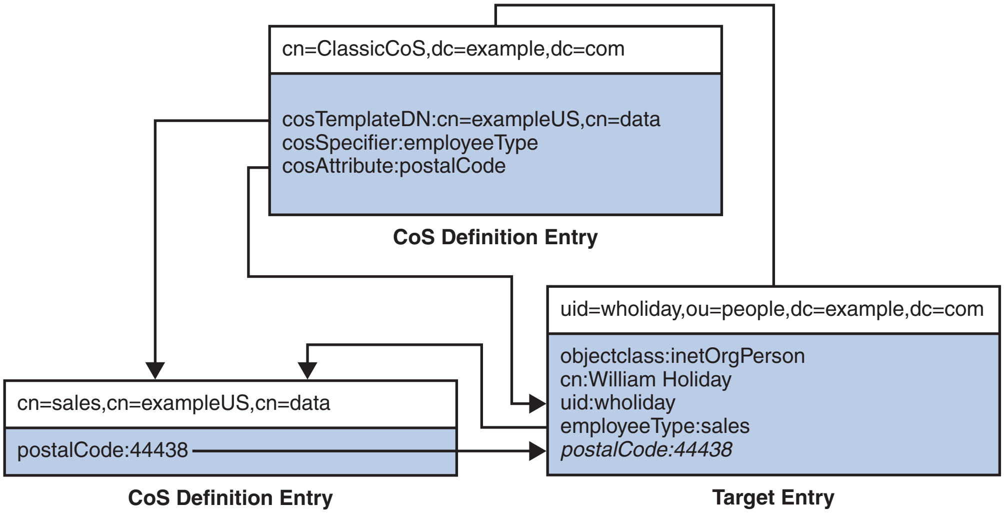

cn=ClassicCoS,dc=example,dc=com
cosTemplateDN:cn=exampleUS,cn=data
cosSpecifier:employeeType
cosAttribute:postalCode
CoS Definition Entry
uid=wholiday,ou=people,dc=example,dc=com
objectclass:inetOrgPerson
cn:William Holiday
uid:wholiday
employeeType:sales
postalCode:44438
cn=sales,cn=exampleUS,cn=data
postalCode:44438
CoS Definition Entry
Target Entry

FIGURE 9–4 Example of a Classic CoS Definition and Template

In this example, the `cosSpecifier` attribute names the `employeeType` attribute. The combination of the `cosSpecifier` attribute and the template DN identifies the template entry as `cn=sales,cn=exampleUS,cn=data`. The template entry then provides the value of the `postalCode` attribute to the target entry.

CoS Priorities

It is possible to create CoS schemes that compete with each other to provide an attribute value. For example, you might have a multivalued `cosSpecifier` in your CoS definition entry. In such a case, you can specify a template priority on each template entry to determine which template provides the attribute value. Set the template priority using the `cosPriority` attribute. This attribute represents the global priority of a particular template numerically. A priority of zero is the highest possible priority.

Templates that contain no `cosPriority` attribute are considered the lowest possible priority. In the case where two or more templates are considered to supply an attribute value and they have the same (or no) priority, a value is chosen arbitrarily. Directory Server can be configured to log messages when it is forced to choose a template arbitrarily.

CoS Limitations

The CoS functionality is a complex mechanism which, for performance and security reasons, is subject to the following limitations:

- Restricted subtrees

  You cannot create CoS definitions in either the `cn=config` or `cn=schema` subtrees.
- Unindexed searches

  Searches in suffixes where an attribute is declared as a CoS-generated attribute will result in an unindexed search. This may have a significant impact on performance. In suffixes where the same attribute is NOT declared as a CoS attribute, the search will be indexed.
- Restricted attribute types

  The following attributes should not be generated by CoS because they do not have the same behavior as real attributes of the same name.
  - `userPassword` - A CoS-generated password value cannot be used to bind to Directory Server.
  - `aci` - Directory Server will not apply any access control based on the contents of a virtual ACI value defined by CoS.
  - `objectclass` - Directory Server will not perform schema checking on the value of a virtual object class defined by CoS.
  - `nsRoleDN` - A CoS-generated `nsRoleDN` value will not be used by the server to generate roles.
- All templates must be local

  The DNs of template entries, either in a CoS definition or in the specifier of the target entry, must refer to local entries in the directory. Templates and the values they contain cannot be retrieved through directory chaining or referrals.
- CoS virtual values cannot be combined with real values

  The values of a CoS attribute are never a combination of real values from the entry and virtual values from the templates. When the CoS overrides a real attribute value, it replaces all real values with those from the templates. However, the CoS mechanism can combine virtual values from several CoS definition entries. For more information, see “CoS Limitations” in the *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.
- Filtered roles cannot use CoS-generated attributes

  The filter string of a filtered role cannot be based on the values of a CoS virtual attribute. However, the specifier attribute in a CoS definition may reference the `nsRole` attribute generated by a role definition. For more information, see “Creating Role-Based Attributes” in the *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.
- Access Control Instructions (ACIs)

The server controls access to attributes generated by a CoS in exactly the same way as regular, stored attributes. However, access control rules that depend on the value of attributes generated by CoS are subject to the conditions described in "CoS Limitations" on page 184.

- CoS cache latency

  The CoS cache is an internal structure that keeps all CoS data in memory to improve performance. This cache is optimized for retrieving CoS data to be used in computing virtual attributes, even while CoS definition and template entries are being updated. Therefore, once definition and template entries have been added or modified, there may be a slight delay before they are taken into account. This delay depends on the number and complexity of CoS definitions, as well as the current server load, but it is usually in the order of a few seconds. Consider this latency before designing overly complex CoS configurations.

CHAPTER 10

Directory Server DSMLv2

For information about DSMLv2 in Directory Server, see the following sections:

Introduction to DSML

Directory Services Markup Language version 2, DSMLv2, is a markup language that describes directory operations in an eXtensible Markup Language (XML) document. For information about the DSMLv2 standard, see *Directory Services Markup Language (DSML) v2.0 [OASIS 200201]* at `http://www.oasis-open.org/specs`.

The complete DSMLv2 specification and supporting documentation can be found at the following locations:

- `http://www.oasis-open.org/committees/dsml/docs/DSMLv2.xsd`
- `http://www.oasis-open.org/committees/dsml/docs/DSMLv2.doc`

Directory Server supports DSMLv2 SOAP over HTTP binding. DSML requests and responses are embedded in the body of SOAP v1.1, and transported in an HTTP/1.1 payload.

The Directory Server Resource Kit contains tools for searching and modifying directories using DSMLv2. See `dsmlsearch`(1) and `dsmlmodify`(1).

By using DSML, non-LDAP clients can perform directory operations. The following figure shows an example deployment where a non-LDAP client makes a requests to modify data on DSML-enabled directory servers.

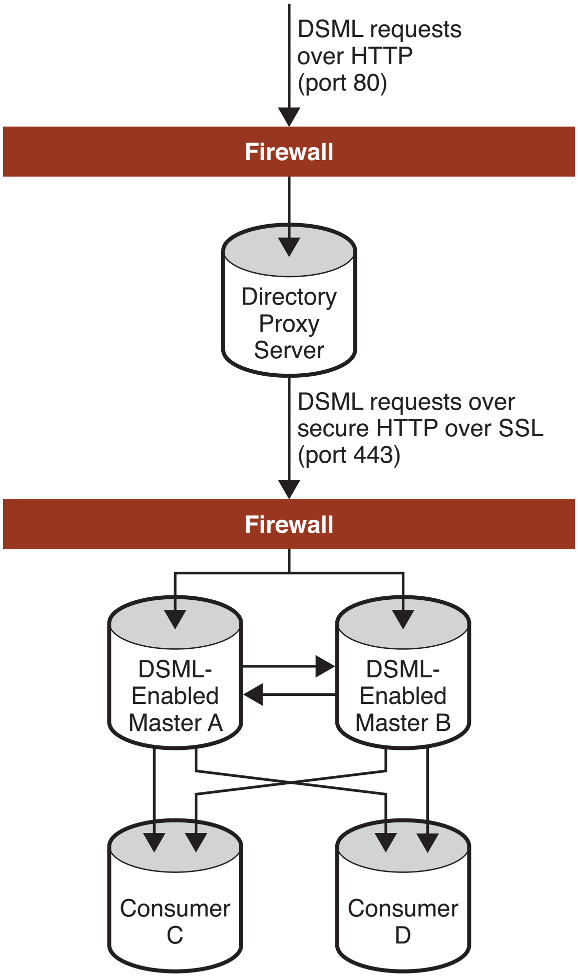


FIGURE 10–1 Sample DSML-Enabled Directory Deployment

In the example deployment, update requests in DSML arrive from non-LDAP client applications cross a firewall over HTTP port 80. The web proxy server enforces the use of secure HTTP over port 443 for the requests to cross a second firewall and enter the intranet domain. The requests are then processed by the two master replicas on Master A and Master B, before being replicated to the non-DSML enabled Consumers C and D.

Implementation of the DSMLv2 Standard

The Directory Server implementation of the DSMLv2 specification has the following restrictions:

Bindings

DSMLv2 defines two normative bindings: a SOAP request/response binding and a file binding that serves as the DSMLv2 analog of LDIF. Directory Server supports the SOAP request/response binding.

Modify DN

Directory Server supports the DSML `modDNRequest` and `modDNResponse` operations.

Abandon request

Directory Server does not support the `abandonRequest` operation, because this operation is of no use over HTTP.

Search operations

Some DSML clients incorrectly send an equality match with value * when a presence match is intended. Directory Server will return zero results from these badly formatted queries. You can detect these incorrect clients by searching for the characters `=\2a` in the access log.

DSML Security

The DSML front end constitutes a restricted HTTP server; it accepts only DSML post operations, it rejects requests that do not conform to the DSMLv2 SOAP binding specifications.

The security of DSML is configured by the following server properties `dsml-client-auth-mode`, `dsml-port`, `dsml-secure-port`, and `dsml-relative-root-url`. For information about these properties, see `server`(5dsconf).

For additional security, consider the following.

- Protect DSML-enabled directory servers by implementing a firewall.
- If you do not impose the use of HTTP over SSL on your clients, implement a demilitarized zone.

DSML Identity Mapping

Identity mapping is required for the following mechanisms: DSML over HTTP, DIGEST-MD5, and GSSAPI SASL. Identity mapping is used to determine a bind DN based on protocol specific credentials provided by the client.

Identity mapping uses the entries in the `cn=identity mapping, cn=config` configuration branch. This branch includes the following containers for the protocols that perform identity mapping:

`cn=HTTP-BASIC, cn=identity mapping, cn=config`
Contains the mappings for DSML-over-HTTP connections.

`cn=DIGEST-MD5, cn=identity mapping, cn=config`
Contains the mappings for client authentication using the DIGEST-MD5 SASL mechanism.

`cn=GSSAPI, cn=identity mapping, cn=config`
Must be created to contain the mappings for client authentication using the GSSAPI SASL mechanism.

A mapping entry defines how to extract credentials about the protocol to use them in a search operation. If a search returns a single user entry, the mapping has succeeded and the connection uses the mapping entry as the bind DN for all operations. If the search returns zero or more than one entry, the mapping fails and the connection does not use the mapping entry as the bind DN.

The protocols that perform identity mapping must have a default mapping. Additionally, The protocols can have any number of custom mappings. The default mapping has the RDN `cn=default`, and custom mappings may have any other RDN that uses `cn` as the naming attribute. All of the custom mappings are evaluated first, in a non deterministic order until one of them succeeds. If all custom mappings fail, the default mapping is applied. If the default mapping fails, authentication of the client fails.

A mapping entry must contain the object classes `top`, `container`, and `dsIdentityMapping`.

The entry can contain the following attributes.

`dsMappedDN:` *DN*
A literal string that defines a DN in the directory. This DN will be used for binding if it exists when the mapping is performed. You may also define the following attributes to perform a search in case this DN does not exist.

`dsSearchBaseDN:` *DN*
The base DN for a search. If omitted, the mapping will search all root suffixes in the entire directory tree, including all naming contexts, but excluding `cn=config`, `cn=monitor`, and `cn=schema`.

`dsSearchScope: base|one|sub`
The scope for a search, either the search base itself, one level of children below the base, or the entire subtree below the base. The default scope for mapping searches is the entire subtree when this attribute is omitted.

`dsSearchFilter:` *filterString*
A filter string to perform the mapping search. LDAP search filters are defined in RFC 4515 on `http://www.ietf.org/rfc/rfc4515.txt`.

Additionally, a mapping entry may also contain the `dsPatternMatching` object class which allows it to use the following attributes:

`dsMatching-pattern:` *patternString*
A string on which to perform pattern matching.

`dsMatching-regexp:` *regularExpression*
A regular expression to apply to the pattern string.

All of the attribute values above, except for `dsSearchScope` may contain placeholders of the format `${`*keyword*`}`, where *keyword* is the name of an element in the protocol-specific credentials. During mapping, the placeholder is substituted for the actual value of the element provided by the client.

After all of the placeholders have been substituted, the pattern matching is performed. The matching pattern is compared to the regular expression, as follows.

- If the regular expression does not match the pattern string, the mapping fails.
- If the regular expression does match the pattern string, the matching values of the regular expression terms in parentheses are available as numbered placeholders for use in other attribute values.

For example, the following mapping could be defined for SASL.

```
dsMatching-pattern: ${Principal}
dsMatching-regexp: (.*)@(.*)\\.(.*)
dsMappedDN: uid=$1,ou=people,dc=$2,dc=$3
```

If a client authenticates with the Principal of `bjensen@example.com`, this mapping will define the following bind DN: `uid=bjensen,ou=people,dc=example,dc=com`. If this DN exists in the directory, the mapping will succeed, the client will be authenticated, and all operations performed during this connection will use this bind DN.

The `dsMatching-pattern` is compared to the `dsMatching-regexp` by using the POSIX `regexec(3C)` and `regcomp(3C)` function calls. Directory Server uses extended regular expressions and all comparisons are case insensitive. For more information, refer to the man pages for these functions.

The attribute values that can contain placeholders must encode any `$`, `{`, and `}` characters that are not part of a placeholder, even if no placeholder is used. You must encode these characters with the following values: `$` as `\\24`, `{` as `\\7B`, and `}` as `\\7D`.

The use of placeholders and substitutions allows you to create mappings that extract a username or any other value from the protocol-specific credentials. The credential can be used to define a mapped DN or perform a search for a corresponding DN anywhere in the directory.

Caution – Creating a poorly defined mapping is a security hole. For example, a mapping to a hard coded DN without pattern matching will always succeed, thereby authenticating clients who might not be directory users. It is safer to define several mappings to handle different client credential formats than to create a single, overly generic and permissive mapping. Always try to map client connections to specific users according to the client credentials.

Content of the HTTP Header

Directory Server supports the HTTP `POST` operation only. The following example shows the minimum fields required to send a DSML request to the server over HTTP:

```
POST /dsml HTTP/1.1
content-length: 450
HOST: hostname
SOAPAction: ""
Content-Type: text/xml
Connection: close
```

The `Connection` field is optional. In HTTP 1.0, the default value of this field is `close`. In HTTP 1.1, however, the default value is `keep-alive`. It is therefore recommended that you include this field with a value of `close` in your last request if you are using HTTP 1.1, to accelerate the dialog.

Additional fields may be included in the HTTP header. If they are supported by Directory Server, their values will override the defaults. If the fields are not supported, the request is not rejected by the server but the fields are ignored.

Accessing the Directory Using DSMLv2

The following examples indicate how to use DSML requests to access and search the directory.

- "An Empty Anonymous DSML Ping Request" on page 193
- "Issuing a DSML Request to Bind as a Particular User" on page 195
- "A DSML Search Request" on page 197

Note that the `content-length:` header in these examples contains the exact length of the DSMLv2 request. For these examples to function correctly, ensure that the editor you use respects these content lengths, or that you modify them accordingly.

An Empty Anonymous DSML Ping Request

The DSML front end is *disabled* by default. For information on how to enable it, refer to "Configuring DSML" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*. To check whether the DSML front end is enabled, send an empty DSML batch request, as shown in Example 10–1.

EXAMPLE 10–1 Empty Anonymous DSML Request

```
POST /dsml HTTP/1.1
 content-length: 451
 HOST: hostname
 SOAPAction: ""
 Content-Type: text/xml
Connection: close
<?xml version=’1.0’ encoding=’UTF-8’?\>
 <soap-env:Envelope
    xmlns:xsd=’http://www.w3.org/2001/XMLSchema’
   xmlns:xsi=’http://www.w3.org/2001/XMLSchema-instance’
   xmlns:soap-env=’http://schemas.xmlsoap.org/soap/envelope/’\>
   <soap-env:Body\>
       <batchRequest
           xmlns=’urn:oasis:names:tc:DSML:2:0:core’          requestID=’Ping!’\>
           <!-- empty batch request --\>
      </batchRequest\>
    </soap-env:Body\>
 </soap-env:Envelope\>
```

The first section of this DSML request contains the HTTP method line, `POST /dsml HTTP/1.1`, followed by a number of HTTP headers. The HTTP method line specifies the HTTP method request and URL to be used by the DSML front end. `POST` is the only HTTP method request accepted by the DSML front end. The `/dsml` URL is the default URL for Directory Server, but can be configured with any other valid URL. The HTTP headers that follow, specify the remaining details of the DSML request.

- `content-length: 451`specifies the exact length of the SOAP/DSML request
- `HOST: hostname`specifies the name of the host Directory Server being contacted.
- `SOAPAction:`is mandatory and informs the directory that you want to perform a DSML request on the HTTP/SOAP stack. It may however, be left empty.
- `Content-Type: text/xml`must have a value of `text/xml` which defines the content as XML.
- `Connection: close`specifies that the connection will be closed once the request has been satisfied. The default HTTP/1.1 behavior is to maintain the connection open.

  The remainder of the request is the SOAP/DSML section. The DSML request begins with the XML prologue header:

```
<?xml version='1.0' encoding='UTF-8'?\>
```

This specifies that the request must be encoded with the UTF-8 character set. The header is followed by the SOAP envelope and body elements that contain the mandatory inclusion of the XML schema, XML schema instance and SOAP name spaces.

The DSML batch request element marks the beginning of the DSML batch request, and is immediately followed by the mandatory inclusion of the DSMLv2 namespace:

```
xmlns='urn:oasis:names:tc:DSML:2:0:core'.
```

The request is optionally identified by the following request ID

```
requestID='Ping!'
```

The empty batch request

```
<!-- empty batch request --\>
```

is XML commented as such, and the SOAP/DSML batch request is closed using the close batch request, close SOAP body, and close SOAP envelope elements.

If the DSML front end is enabled, an empty DSML response is returned, as shown in Example 10–2.

EXAMPLE 10–2 Empty Anonymous DSML Response

```
HTTP/1.1 200 OK
Cache-control: no-cache
Connection: close
Date: Mon, 11 Dec 2006 13:56:49 GMT
Accept-Ranges: none
Server: Sun-Java(tm)-System-Directory/6.3
Content-Type: text/xml; charset="utf-8"
Content-Length: 500
<?xml version='1.0' encoding='UTF-8' ?\>
<soap-env:Envelope
   xmlns:xsd='http://www.w3.org/2001/XMLSchema'
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns:soap-env='http://schemas.xmlsoap.org/soap/envelope/'
   \>
<soap-env:Body\>
<batchResponse
   xmlns:xsd='http://www.w3.org/2001/XMLSchema'
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns='urn:oasis:names:tc:DSML:2:0:core'
   requestID='Ping!'
   \>
```

EXAMPLE 10–2 Empty Anonymous DSML Response *(Continued)*

```
</batchResponse\>
</soap-env:Body\>
</soap-env:Envelope\>
```

If nothing is returned, you can conclude that the front end is disabled.

Maximum limits exist for the number of clients connecting simultaneously to the directory and for the size of the DSML requests. The limit for the number of clients is specified by the `dsml-max-parser-count` and `dsml-min-parser-count` server properties and the request size limit by the server property `dsml-request-max-size`. See `server`(5dsconf).

Issuing a DSML Request to Bind as a Particular User

To issue a DSML request you can bind to the directory as a specified user or anonymously. To bind as a specified user, the request must include an HTTP authorization header containing a UID and a password that are mapped to a DN, as shown in Example 10–3.

EXAMPLE 10–3 DSML Extended Operation: Bind as a Particular User

```
POST /dsml HTTP/1.1
content-length: 578
content-Type: text/xml; charset="utf-8"
HOST: hostname
Authorization: Basic ZWFzdGVyOmVnZw==
SOAPAction: ""
Connection: close
<?xml version=’1.0’ encoding=’UTF-8’?\>
<soap-env:Envelope
   xmlns:xsd=’http://www.w3.org/2001/XMLSchema’
   xmlns:xsi=’http://www.w3.org/2001/XMLSchema-instance’
   xmlns:soap-env=’http://schemas.xmlsoap.org/soap/envelope/’\>
   <soap-env:Body\>
     <batchRequest
        xmlns=’urn:oasis:names:tc:DSML:2:0:core’\>
        <extendedRequest\>
          <requestName\>1.3.6.1.4.1.4203.1.11.3</requestName\>
        </extendedRequest\>
     </batchRequest\>
   </soap-env:Body\>
</soap-env:Envelope\>
```

In this example the HTTP authorization header transports the user ID `easter` and the password `egg`, which, in clear, appears as `easter:egg`, and encoded in base64 as `Authorization: Basic ZWFzdGVyOmVnZw==`.

The <extendedRequest\> tag is used to specify an LDAP Extended Operation. The <requestName\> tag is used to specify the OID of the extended operation. In this example, the OID 1.3.6.1.4.1.4203.1.11.3 identifies the whoami extended operation.

The response to the DSML extended operation shows the DN of the user that made the bind request. In Example 10–4, the whoami response, which contains the DN, is shown in the response line.

```
<response\>dn:uid=easter,ou=people,dc=example,dc=com</response\>
```

EXAMPLE 10–4 Response to DSML Extended Operation

```
HTTP/1.1 200 OK
Cache-control: no-cache
Connection: close
Date: Fri, 15 Dec 2006 09:15:09 GMT
Accept-Ranges: none
Server: Sun-Java(tm)-System-Directory/6.3
Content-Type: text/xml; charset="utf-8"
Content-Length: 697

<?xml version='1.0' encoding='UTF-8' ?\>
<soap-env:Envelope
   xmlns:xsd='http://www.w3.org/2001/XMLSchema'
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns:soap-env='http://schemas.xmlsoap.org/soap/envelope/'
   \>
<soap-env:Body\>
<batchResponse
   xmlns:xsd='http://www.w3.org/2001/XMLSchema'
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns='urn:oasis:names:tc:DSML:2:0:core'
   \>
   <extendedResponse\>
      <resultCode code='0' descr='success'/\>
   <responseName\>1.3.6.1.4.1.4203.1.11.3</responseName\>
   <response\>dn:uid=easter,ou=people,dc=example,dc=com</response\>
   </extendedResponse\>
</batchResponse\>
</soap-env:Body\>
</soap-env:Envelope\>
```

For anonymous access, no HTTP authorization header is required, although anonymous access is often subject to strict access controls, and possibly to data access restrictions. Similarly, you can issue DSML requests to perform LDAP operations by LDAP proxy.

Because DSML requests are managed on a batch basis, if you issue requests by LDAP proxy, the required DSML proxy authorization request must be the first in a given batch of requests.

A DSML Search Request

Example 10–5 shows a DSML base object search request on the root DSE entry.

EXAMPLE 10–5 DSML Search Request

```
POST /dsml HTTP/1.1
HOST: hostname
Content-Length: 1081
Content-Type: text/xml
SOAPAction: ""
Connection: close
<?xml version=’1.0’ encoding=’UTF-8’?\>
<soap-env:Envelope
   xmlns:xsd=’http://www.w3.org/2001/XMLSchema’
   xmlns:xsi=’http://www.w3.org/2001/XMLSchema-instance’
   xmlns:soap-env=’http://schemas.xmlsoap.org/soap/envelope/’
   \>
   <soap-env:Body\>
      <batchRequest
        xmlns=’urn:oasis:names:tc:DSML:2:0:core’
        requestID=’Batch of search requests’
        \>
        <searchRequest
            dn=""
            requestID="search on Root DSE"
            scope="baseObject"
            derefAliases="neverDerefAliases"
            typesOnly="false"
            \>
            <filter\>
               <present name="objectClass"/\>
            </filter\>
            <attributes\>
               <attribute name="namingContexts"/\>
               <attribute name="supportedLDAPversion"/\>
               <attribute name="vendorName"/\>
               <attribute name="vendorVersion"/\>
               <attribute name="supportedSASLMechanisms"/\>
            </attributes\>
        </searchRequest\>
      </batchRequest\>
   </soap-env:Body\>
</soap-env:Envelope\>
```

- `dn=""requestID="search on Root DSE"`specifies that the search operation requests data under the root DSE entry (empty DN) and is identified with an optional request ID attribute.

- `scope="baseObject"`specifies that the search is a base object search.
- `derefAliases="neverDerefAliases"`specifies that the aliases should not be dereferenced while searching or locating the base object of the search. This is the only `derefAliases` value supported by Directory Server.
- `typesOnly="false"`specifies that both the attribute names and their values be returned. `typesOnly="true"` would return attribute names only. The default value for this attribute is false.

  For the entry to match the filter, the presence of `objectclass` filter is used as follows.

  ```
  <filter\>
     <present name="objectClass"/\>
  </filter\>
  ```

  This is equivalent to the LDAP filter string `(objectclass=*)`. The filter is followed by the list of desired attributes.

  ```
  <attributes\>
     <attribute name="namingContexts"/\>
     <attribute name="supportedLDAPversion"/\>
     <attribute name="vendorName"/\>
     <attribute name="vendorVersion"/\>
     <attribute name="supportedSASLMechanisms"/\>
  </attributes\>
  ```

EXAMPLE 10–6 DSML Search Response

```
HTTP/1.1 200 OK
Cache-control: no-cache
Connection: close
Date: Fri, 15 Dec 2006 09:21:43 GMT
Accept-Ranges: none
Server: Sun-Java(tm)-System-Directory/6.3
Content-Type: text/xml; charset="utf-8"
Content-Length: 1287

<?xml version='1.0' encoding='UTF-8' ?\>
<soap-env:Envelope
   xmlns:xsd='http://www.w3.org/2001/XMLSchema'
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns:soap-env='http://schemas.xmlsoap.org/soap/envelope/'
   \>
<soap-env:Body\>
<batchResponse
   xmlns:xsd='http://www.w3.org/2001/XMLSchema'
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns='urn:oasis:names:tc:DSML:2:0:core'
```

EXAMPLE 10–6 DSML Search Response *(Continued)*

```
   requestID=’Batch of search requests’
   \>
   <searchResponse requestID=’search on Root DSE’\>
   <searchResultEntry\>
      <attr name=’namingContexts’\>
      <value\>dc=example,dc=com</value\>
      </attr\>
      <attr name=’supportedLDAPVersion’\>
      <value\>2</value\>
      <value\>3</value\>
      </attr\>
      <attr name=’vendorName’\>
      <value\>Sun Microsystems, Inc.</value\>
      </attr\>
      <attr name=’vendorVersion’\>
      <value\>Sun-Java(tm)-System-Directory/6.3</value\>
      </attr\>
      <attr name=’supportedSASLMechanisms’\>
      <value\>EXTERNAL</value\>
      <value\>GSSAPI</value\>
      <value\>DIGEST-MD5</value\>
      </attr\>
   </searchResultEntry\>
   <searchResultDone\>
      <resultCode code=’0’ descr=’success’/\>
   </searchResultDone\>
   </searchResponse\>
</batchResponse\>
</soap-env:Body\>
</soap-env:Envelope\>
```

CHAPTER 11

Directory Server Internationalization Support

Directory Server provides support for storing, managing, and searching for entries and their associated attributes in different languages.

Data inside the internationalized directory is stored in UTF-8 format. Therefore, Directory Server supports all international characters by default. The internationalized directory can be used to specify matching rules and collation orders based on language preferences in search operations. For information about the internationalized directory, see the following sections:

- "About Locales" on page 201
- "Identifying Supported Locales" on page 202
- "Supported Language Subtypes" on page 208

About Locales

A locale identifies language-specific information about how users in a specific region, culture, or custom expect data to be presented. Locales define how data in different languages is interpreted, sorted, and collated.Directory Server supports multiple languages through the use of locales.

A locale specifies the following information.

Code page

The code page is an internal table used by an operating system to relate keyboard keys to character fonts displayed on a screen. A locale can indicate what code page an application should select for interaction with an end user.

Collation order

The collation order provides information about how the characters of a given language should be sorted. The collation order specifies the following information:

- The sequence of the letters in the alphabet
- How to compare letters with accents to letters without accents

- Whether there are characters that can be ignored when comparing strings
- The direction, left to right, right to left, or up and down, in which the language is read

Character type

The character type distinguishes alphabetic characters from numeric or other characters. It defines the mapping of uppercase letters to lowercase letters. For example, in some languages, the pipe character (|) is considered punctuation, while in other languages it is considered as alphabetic.

Monetary format

The monetary format specifies the following information: the monetary symbol used in a region, whether the symbol goes before or after its value, and how monetary units are represented.

Time and date formats

The time and date formats determine the appearance of times and dates in a region. The time format indicates whether the locale uses a 12–hour clock or 24-hour clock. The date format includes both the short date order and the long date format, and include the names of months and days of the week in each language.

Identifying Supported Locales

When you perform directory operations that require you to specify a locale, such as a search operation, you can use a language tag or a collation order object identifier, OID.

A language tag is a string that begins with the two-character lowercase language code that identifies the language, as defined in ISO standard 639. If necessary to distinguish regional differences in language, the language tag may also contain a country code, which is a two-character string, as defined in ISO standard 3166. The language code and country code are separated by a hyphen. For example, the language tag used to identify the American English locale is `en-US`.

An OID is a decimal number that uniquely identifies an object, such as an attribute or object class.

When you perform an international search in a directory, use either the language tag or the OID to identify the collation order you want to use. When you set up an international index, use the OIDs.

The following table lists the locales supported by Directory Server. It identifies the associated language tags and OIDs.

TABLE 11–1 Supported Locales

Locale	Tag	Collation Order OID	Backward Compatible OID
Afrikaans	af	1.3.6.1.4.1.42.2.27.9.4.1.1	
Amharic Ethiopia	am	1.3.6.1.4.1.42.2.27.9.4.2.1	
Arabic	ar	1.3.6.1.4.1.42.2.27.9.4.3.1	2.16.840.1.113730.3.3.2.1.1
Arabic United Arab Emirates	ar-AE	1.3.6.1.4.1.42.2.27.9.4.4.1	
Arabic Bahrain	ar-BH	1.3.6.1.4.1.42.2.27.9.4.5.1	
Arabic Algeria	ar-DZ	1.3.6.1.4.1.42.2.27.9.4.6.1	
Arabic Egypt	ar-EG	1.3.6.1.4.1.42.2.27.9.4.7.1	
Arabic India	ar-IN	1.3.6.1.4.1.42.2.27.9.4.8.1	
Arabic Iraq	ar-IQ	1.3.6.1.4.1.42.2.27.9.4.9.1	
Arabic Jordan	ar-JO	1.3.6.1.4.1.42.2.27.9.4.10.1	
Arabic Kuwait	ar-KW	1.3.6.1.4.1.42.2.27.9.4.11.1	
Arabic Lebanon	ar-LB	1.3.6.1.4.1.42.2.27.9.4.12.1	
Arabic Libya	ar-LY	1.3.6.1.4.1.42.2.27.9.4.13.1	
Arabic Morocco	ar-MA	1.3.6.1.4.1.42.2.27.9.4.14.1	
Arabic Oman	ar-OM	1.3.6.1.4.1.42.2.27.9.4.15.1	
Arabic Qatar	ar-QA	1.3.6.1.4.1.42.2.27.9.4.16.1	
Arabic Saudi Arabia	ar-SA	1.3.6.1.4.1.42.2.27.9.4.17.1	
Arabic Sudan	ar-SD	1.3.6.1.4.1.42.2.27.9.4.18.1	
Arabic Syria	ar-SY	1.3.6.1.4.1.42.2.27.9.4.19.1	
Arabic Tunisia	ar-TN	1.3.6.1.4.1.42.2.27.9.4.20.1	
Arabic Yemen	ar-YE	1.3.6.1.4.1.42.2.27.9.4.21.1	
Byelorussian	be	1.3.6.1.4.1.42.2.27.9.4.22.1	2.16.840.1.113730.3.3.2.2.1
Bulgarian	bg	1.3.6.1.4.1.42.2.27.9.4.23.1	2.16.840.1.113730.3.3.2.3.1
Bengali India	bn	1.3.6.1.4.1.42.2.27.9.4.24.1	
Catalan	ca	1.3.6.1.4.1.42.2.27.9.4.25.1	2.16.840.1.113730.3.3.2.4.1
Czech	cs	1.3.6.1.4.1.42.2.27.9.4.26.1	2.16.840.1.113730.3.3.2.5.1
Danish	da	1.3.6.1.4.1.42.2.27.9.4.27.1	2.16.840.1.113730.3.3.2.6.1

TABLE 11–1 Supported Locales *(Continued)*

Locale	Tag	Collation Order OID	Backward Compatible OID
German	de or de-DE	1.3.6.1.4.1.42.2.27.9.4.28.1	2.16.840.1.113730.3.3.2.7.1
German Austria	de-AT	1.3.6.1.4.1.42.2.27.9.4.29.1	2.16.840.1.113730.3.3.2.8.1
German Belgium	de-BE	1.3.6.1.4.1.42.2.27.9.4.30.1	
German Swiss	de-CH	1.3.6.1.4.1.42.2.27.9.4.31.1	2.16.840.1.113730.3.3.2.9.1
German Luxembourg	de-LU	1.3.6.1.4.1.42.2.27.9.4.32.1	
Greek	el	1.3.6.1.4.1.42.2.27.9.4.33.1	2.16.840.1.113730.3.3.2.10.1
English (US)	en-US	1.3.6.1.4.1.42.2.27.9.4.34.1	2.16.840.1.113730.3.3.2.11.1
English Australian	en-AU	1.3.6.1.4.1.42.2.27.9.4.35.1	
English Canada	en-CA	1.3.6.1.4.1.42.2.27.9.4.36.1	2.16.840.1.113730.3.3.2.12.1
English Great Britain	en-GB	1.3.6.1.4.1.42.2.27.9.4.37.1	2.16.840.1.113730.3.3.2.13.1
English Hong Kong	en-HK	1.3.6.1.4.1.42.2.27.9.4.38.1	
English Ireland	en-IE	1.3.6.1.4.1.42.2.27.9.4.39.1	2.16.840.1.113730.3.3.2.14.1
English India	en-IN	1.3.6.1.4.1.42.2.27.9.4.40.1	
English Malta	en-MT	1.3.6.1.4.1.42.2.27.9.4.41.1	
English New Zealand	en-NZ	1.3.6.1.4.1.42.2.27.9.4.42.1	
English Philippines	en-PH	1.3.6.1.4.1.42.2.27.9.4.43.1	
English Singapore	en-SG	1.3.6.1.4.1.42.2.27.9.4.44.1	
English Virgin Island	en-VI	1.3.6.1.4.1.42.2.27.9.4.45.1	
English South Africa	en-ZA	1.3.6.1.4.1.42.2.27.9.4.46.1	
English Zimbabwe	en-ZW	1.3.6.1.4.1.42.2.27.9.4.47.1	
Esperanto	eo	1.3.6.1.4.1.42.2.27.9.4.48.1	
Spanish	es or es-ES	1.3.6.1.4.1.42.2.27.9.4.49.1	2.16.840.1.113730.3.3.2.15.1
Spanish Argentina	es-AR	1.3.6.1.4.1.42.2.27.9.4.50.1	
Spanish Bolivia	es-BO	1.3.6.1.4.1.42.2.27.9.4.51.1	
Spanish Chile	es-CL	1.3.6.1.4.1.42.2.27.9.4.52.1	
Spanish Colombia	es-CO	1.3.6.1.4.1.42.2.27.9.4.53.1	

TABLE 11–1 Supported Locales *(Continued)*

Locale	Tag	Collation Order OID	Backward Compatible OID
Spanish Costa Rica	es-CR	1.3.6.1.4.1.42.2.27.9.4.54.1	
Spanish Dominican Rep.	es-DO	1.3.6.1.4.1.42.2.27.9.4.55.1	
Spanish Ecuador	es-EC	1.3.6.1.4.1.42.2.27.9.4.56.1	
Spanish Guatemala	es-GT	1.3.6.1.4.1.42.2.27.9.4.57.1	
Spanish Honduras	es-HN	1.3.6.1.4.1.42.2.27.9.4.58.1	
Spanish Mexico	es-MX	1.3.6.1.4.1.42.2.27.9.4.59.1	
Spanish Nicaragua	es-NI	1.3.6.1.4.1.42.2.27.9.4.60.1	
Spanish Panama	es-PA	1.3.6.1.4.1.42.2.27.9.4.61.1	
Spanish Peru	es-PE	1.3.6.1.4.1.42.2.27.9.4.62.1	
Spanish Puerto Rico	es-PR	1.3.6.1.4.1.42.2.27.9.4.63.1	
Spanish Paraguay	es-PY	1.3.6.1.4.1.42.2.27.9.4.64.1	
Spanish El Salvador	es-SV	1.3.6.1.4.1.42.2.27.9.4.65.1	
Spanish US	es-US	1.3.6.1.4.1.42.2.27.9.4.66.1	
Spanish Uruguay	es-UY	1.3.6.1.4.1.42.2.27.9.4.67.1	
Spanish Venezuela	es-VE	1.3.6.1.4.1.42.2.27.9.4.68.1	
Estonian	et	1.3.6.1.4.1.42.2.27.9.4.69.1	2.16.840.1.113730.3.3.2.16.1
Basque	eu	1.3.6.1.4.1.42.2.27.9.4.70.1	
Persian	fa	1.3.6.1.4.1.42.2.27.9.4.71.1	
Persian India	fa-IN	1.3.6.1.4.1.42.2.27.9.4.72.1	
Persian Iran	fa-IR	1.3.6.1.4.1.42.2.27.9.4.73.1	
Finnish	fi	1.3.6.1.4.1.42.2.27.9.4.74.1	2.16.840.1.113730.3.3.2.17.1
Faeroese	fo	1.3.6.1.4.1.42.2.27.9.4.75.1	
French	fr or fr-FR	1.3.6.1.4.1.42.2.27.9.4.76.1	2.16.840.1.113730.3.3.2.18.1
French Belgium	fr-BE	1.3.6.1.4.1.42.2.27.9.4.77.1	2.16.840.1.113730.3.3.2.19.1
French Canada	fr-CA	1.3.6.1.4.1.42.2.27.9.4.78.1	2.16.840.1.113730.3.3.2.20.1
French Swiss	fr-CH	1.3.6.1.4.1.42.2.27.9.4.79.1	2.16.840.1.113730.3.3.2.21.1
French Luxembourg	fr-LU	1.3.6.1.4.1.42.2.27.9.4.80.1	
Irish	ga	1.3.6.1.4.1.42.2.27.9.4.81.1	

TABLE 11–1 Supported Locales *(Continued)*

Locale	Tag	Collation Order OID	Backward Compatible OID
Galician	gl	1.3.6.1.4.1.42.2.27.9.4.82.1	
Gujarati	gu	1.3.6.1.4.1.42.2.27.9.4.83.1	
Manx Gaelic (Isle of Man)	gv	1.3.6.1.4.1.42.2.27.9.4.84.1	
Hebrew	he or iw	1.3.6.1.4.1.42.2.27.9.4.85.1	2.16.840.1.113730.3.3.2.27.1
Hindi	hi	1.3.6.1.4.1.42.2.27.9.4.86.1	
Croatian	hr	1.3.6.1.4.1.42.2.27.9.4.87.1	2.16.840.1.113730.3.3.2.22.1
Hungarian	hu	1.3.6.1.4.1.42.2.27.9.4.88.1	2.16.840.1.113730.3.3.2.23.1
Armenian	hy	1.3.6.1.4.1.42.2.27.9.4.89.1	
Indonesian	id	1.3.6.1.4.1.42.2.27.9.4.90.1	
Icelandic	is	1.3.6.1.4.1.42.2.27.9.4.91.1	2.16.840.1.113730.3.3.2.24.1
Italian	it	1.3.6.1.4.1.42.2.27.9.4.92.1	2.16.840.1.113730.3.3.2.25.1
Italian Swiss	it-CH	1.3.6.1.4.1.42.2.27.9.4.93.1	2.16.840.1.113730.3.3.2.26.1
Japanese	ja	1.3.6.1.4.1.42.2.27.9.4.94.1	2.16.840.1.113730.3.3.2.28.1
Greenlandic	kl	1.3.6.1.4.1.42.2.27.9.4.95.1	
Kannada	kn	1.3.6.1.4.1.42.2.27.9.4.96.1	
Korean	ko	1.3.6.1.4.1.42.2.27.9.4.97.1	2.16.840.1.113730.3.3.2.29.1
Konkani	kok	1.3.6.1.4.1.42.2.27.9.4.98.1	
Cornish	kw	1.3.6.1.4.1.42.2.27.9.4.99.1	
Lithuanian	lt	1.3.6.1.4.1.42.2.27.9.4.100.1	2.16.840.1.113730.3.3.2.30.1
Latvian or Lettish	lv	1.3.6.1.4.1.42.2.27.9.4.101.1	2.16.840.1.113730.3.3.2.31.1
Macedonian	mk	1.3.6.1.4.1.42.2.27.9.4.102.1	2.16.840.1.113730.3.3.2.32.1
Marathi	mr	1.3.6.1.4.1.42.2.27.9.4.103.1	
Maltese	mt	1.3.6.1.4.1.42.2.27.9.4.104.1	
Dutch	nl or nl-NL	1.3.6.1.4.1.42.2.27.9.4.105.1	2.16.840.1.113730.3.3.2.33.1
Dutch Belgium	nl-BE	1.3.6.1.4.1.42.2.27.9.4.106.1	2.16.840.1.113730.3.3.2.34.1
Norwegian	no or no-NO	1.3.6.1.4.1.42.2.27.9.4.107.1	2.16.840.1.113730.3.3.2.35.1

TABLE 11–1 Supported Locales *(Continued)*

Locale	Tag	Collation Order OID	Backward Compatible OID
Norwegian Nynorsk	no-NO-NY	1.3.6.1.4.1.42.2.27.9.4.108.1	2.16.840.1.113730.3.3.2.37.1
Norwegian Nynorsk	nn	1.3.6.1.4.1.42.2.27.9.4.109.1	
Norwegian Bokmål	nb or no-NO-B	1.3.6.1.4.1.42.2.27.9.4.110.1	2.16.840.1.113730.3.3.2.36.1
Oromo (Afan)	om	1.3.6.1.4.1.42.2.27.9.4.111.1	
Oromo Ethiopia	om-ET	1.3.6.1.4.1.42.2.27.9.4.112.1	
Oromo Kenya	om-KE	1.3.6.1.4.1.42.2.27.9.4.113.1	
Polish	pl	1.3.6.1.4.1.42.2.27.9.4.114.1	2.16.840.1.113730.3.3.2.38.1
Portuguese	pt or pt-PT	1.3.6.1.4.1.42.2.27.9.4.115.1	
Portuguese Brazil	pt-BR	1.3.6.1.4.1.42.2.27.9.4.116.1	
Romanian	ro	1.3.6.1.4.1.42.2.27.9.4.117.1	2.16.840.1.113730.3.3.2.39.1
Russian	ru or ru-RU	1.3.6.1.4.1.42.2.27.9.4.118.1	2.16.840.1.113730.3.3.2.40.1
Russian Ukraine	ru-UA	1.3.6.1.4.1.42.2.27.9.4.119.1	
Serbo-Croatian	sh	1.3.6.1.4.1.42.2.27.9.4.120.1	2.16.840.1.113730.3.3.2.41.1
Slovak	sk	1.3.6.1.4.1.42.2.27.9.4.121.1	2.16.840.1.113730.3.3.2.42.1
Slovenian	sl	1.3.6.1.4.1.42.2.27.9.4.122.1	2.16.840.1.113730.3.3.2.43.1
Somali	so or so-SO	1.3.6.1.4.1.42.2.27.9.4.123.1	
Somali Djibouti	so-DJ	1.3.6.1.4.1.42.2.27.9.4.124.1	
Somali Ethiopia	so-ET	1.3.6.1.4.1.42.2.27.9.4.125.1	
Somali Kenya	so-KE	1.3.6.1.4.1.42.2.27.9.4.126.1	
Albanian	sq	1.3.6.1.4.1.42.2.27.9.4.127.1	2.16.840.1.113730.3.3.2.44.1
Serbian	sr	1.3.6.1.4.1.42.2.27.9.4.128.1	2.16.840.1.113730.3.3.2.45.1
Swedish	sv-SE	1.3.6.1.4.1.42.2.27.9.4.129.1	2.16.840.1.113730.3.3.2.46.1
Swedish Finland	sv-FI	1.3.6.1.4.1.42.2.27.9.4.130.1	
Swahili	sw	1.3.6.1.4.1.42.2.27.9.4.131.1	
Swahili Kenya	sw-KE	1.3.6.1.4.1.42.2.27.9.4.132.1	

TABLE 11–1 Supported Locales *(Continued)*

Locale	Tag	Collation Order OID	Backward Compatible OID
Swahili Tanzania	sw-TZ	1.3.6.1.4.1.42.2.27.9.4.133.1	
Tamil	ta	1.3.6.1.4.1.42.2.27.9.4.134.1	
Telugu	te	1.3.6.1.4.1.42.2.27.9.4.135.1	
Thai	th	1.3.6.1.4.1.42.2.27.9.4.136.1	
Tigrinya	ti	1.3.6.1.4.1.42.2.27.9.4.137.1	
Tigrinya Eritrea	ti-ER	1.3.6.1.4.1.42.2.27.9.4.138.1	
Tigrinya Ethiopia	ti-ET	1.3.6.1.4.1.42.2.27.9.4.139.1	
Turkish	tr	1.3.6.1.4.1.42.2.27.9.4.140.1	2.16.840.1.113730.3.3.2.47.1
Ukrainian	uk	1.3.6.1.4.1.42.2.27.9.4.141.1	2.16.840.1.113730.3.3.2.48.1
Vietnamese	vi	1.3.6.1.4.1.42.2.27.9.4.142.1	
Chinese	zh	1.3.6.1.4.1.42.2.27.9.4.143.1	2.16.840.1.113730.3.3.2.49.1
Chinese China	zh-CN	1.3.6.1.4.1.42.2.27.9.4.144.1	
Chinese Hong Kong	zh-HK	1.3.6.1.4.1.42.2.27.9.4.145.1	
Chinese Mongolia	zh-MO	1.3.6.1.4.1.42.2.27.9.4.146.1	
Chinese Singapore	zh-SG	1.3.6.1.4.1.42.2.27.9.4.147.1	
Chinese Taiwan	zh-TW	1.3.6.1.4.1.42.2.27.9.4.148.1	2.16.840.1.113730.3.3.2.50.1

Supported Language Subtypes

Language subtypes can be used by clients to indicate specific attributes in characters of a language other than the default language of a deployment. For example, German users may prefer to see addresses in German when possible. In this case, you can select German as a language subtype for the `streetAddress` attribute so that users can search for either the English or the German representation of the address. If you specify a language subtype for an attribute, the subtype is added to the attribute name as follows:`attribute;lang-subtype`.

The following listing shows an English language and German language subtype for the `streetAddress` attribute:

```
streetAddress;lang-en: 10 Schlossplatz, 76113, Karlsruhe, Germany
streetAddress;lang-de: Schloßplatz 10, 76113, Karlsruhe, Deutschland
```

The following table contains the list of supported language subtypes.

TABLE 11–2 Supported Language Subtypes

Language	Language Tag
Afrikaans	af
Albanian	sq
Amharic Ethiopia	am
Arabic	ar
Armenian	hy
Basque	eu
Bengali India	bn
Bulgarian	bg
Byelorussian	be
Catalan	ca
Chinese	zh
Cornish	kw
Croatian	hr
Czech	cs
Danish	da
Dutch	nl
English	en
Esperanto	eo
Estonian	et
Faeroese	fo
Finnish	fi
French	fr
Galician	gl
German	de
Greek	el
Greenlandic	kl
Gujarati	gu

TABLE 11–2 Supported Language Subtypes *(Continued)*

Language	Language Tag
Hebrew	he or iw
Hindi	hi
Hungarian	hu
Icelandic	is
Indonesian	id
Irish	ga
Italian	it
Japanese	ja
Kannada	kn
Konkani	kok
Korean	ko
Latvian or Lettish	lv
Lithuanian	lt
Macedonian	mk
Maltese	mt
Manx (Isle of Man)	gv
Marathi	mr
Norwegian	no
Oromo	om
Persian	fa
Polish	pl
Portuguese	pt
Romanian	ro
Russian	ru
Serbian	sr
Serbo-Croatian	sh
Slovak	sk
Slovenian	sl

TABLE 11–2 Supported Language Subtypes *(Continued)*

Language	Language Tag
Somali	so
Spanish	es
Swahili	sw
Swedish	sv
Tamil	ta
Telugu	te
Thai	th
Tigrinya	ti
Turkish	tr
Ukrainian	uk
Vietnamese	vi

CHAPTER 12

Directory Server LDAP URLs

One way to express an LDAP query is to use a URL to specify the Directory Server host machine and the DN or filter for the search. Directory Server responds to queries sent as LDAP URLs and returns an HTML page representing the results. In this way, if anonymous searching is permitted, web browsers can perform searches of the directory. You can also use LDAP URLs to specify target entries when you manage Directory Server referrals or when you access control instructions.

For information about LDAP URLs, see the following sections:

- "Components of an LDAP URL" on page 213
- "Escaping Unsafe Characters" on page 215
- "Examples of LDAP URLs" on page 215

Components of an LDAP URL

LDAP URLs have the following syntax:

`ldap[s]://`*hostname*`:`*port*`/`*base_dn*`?`*attributes*`?`*scope*`?`*filter*

When `ldap://` is specified, standard LDAP is used to connect to the LDAP servers. When `ldaps://` is specified, LDAP over SSL is used to connect to the LDAP server.

TABLE 12–1 LDAP URL Components

Component	Description
hostname	Name (or IP address in dotted format) of the LDAP server. For example: `ldap.example.com` or `192.168.1.100`

TABLE 12–1 LDAP URL Components *(Continued)*

Component	Description
port	Port number of the LDAP server. If no port is specified, the standard LDAP port (389) or LDAPS port (636) is used.
base_dn	Distinguished name (DN) of an entry in the directory. This DN identifies the entry that is the starting point of the search. If no base DN is specified, the search starts at the root of the directory tree.
attributes	The attributes to be returned. To specify more than one attribute, use commas to separate the attributes. For example, "`cn,mail,telephoneNumber`". If no attributes are specified in the URL, all attributes are returned.
scope	The scope of the search. The scope can be one of these values: ■ `base` retrieves information about the distinguished name (*base_dn*) specified in the URL only. ■ `one` retrieves information about entries one level below the distinguished name (*base_dn*) specified in the URL. The base entry is not included in this scope. ■ `sub` retrieves information about entries at all levels below the distinguished name (*base_dn*) specified in the URL. The base entry is included in this scope. If no scope is specified, the server performs a `base` search.
filter	Search filter to apply to entries within the specified scope of the search. If no filter is specified, the server uses the filter `objectClass=*`.

The following components are identified by their positions in the URL: `attributes`, `scope`, and `filter` are. If you do not want to specify a component, you must include a question mark to delimit the field. Two consecutive question marks, `??`, indicate that no attributes have been specified.

For example, to specify a subtree search starting from "`dc=example,dc=com`" that returns all attributes for entries matching "`(sn=Jensen)`", use the following LDAP URL.

```
ldap://ldap.example.com/dc=example,dc=com??sub?(sn=Jensen)
```

Because no specific attributes are identified in the URL, all attributes are returned in the search.

Escaping Unsafe Characters

Unsafe characters in a URL must be represented by a special sequence of characters. The following table lists the characters that are unsafe within URLs, and provides the associated escape characters to use in place of the unsafe character.

TABLE 12–2 Characters That Are Unsafe Within URLs

Unsafe Character	Escape Characters
space	%20
<	%3c
\>	%3e
"	%22
#	%23
%	%25
{	%7b
}	%7d
|	%7c
\\	%5c
^	%5e
~	%7e
[	%5b
]	%5d
”	%60

Examples of LDAP URLs

The syntax for LDAP URLs does not include any means for specifying credentials or passwords. Search request initiated through LDAP URLs are unauthenticated (anonymous), unless the LDAP client that supports LDAP URLs provides an authentication mechanism. This section gives examples of LDAP URLs.

EXAMPLE 12–1 Base Search for an Entry

The following LDAP URL specifies a base search for the entry with the distinguished name `dc=example,dc=com`.

EXAMPLE 12–1 Base Search for an Entry *(Continued)*

```
ldap://ldap.example.com/dc=example,dc=com
```

- Because no port number is specified, the standard LDAP port number 389 is used.
- Because no attributes are specified, the search returns all attributes.
- Because no search scope is specified, the search is restricted to the base entry `dc=example,dc=com`.
- Because no filter is specified, the directory uses the default filter `objectclass=*`.

EXAMPLE 12–2 Retrieving `postalAddress` Attribute of an Entry

The following LDAP URL retrieves the `postalAddress` attribute of the entry with the DN `dc=example,dc=com`:

```
ldap://ldap.example.com/dc=example,dc=com?postalAddress
```

- Because no search scope is specified, the search is restricted to the base entry `dc=example,dc=com`.
- Because no filter is specified, the directory uses the default filter `objectclass=*`.

EXAMPLE 12–3 Retrieving `cn` and `mail` Attributes of an Entry

The following LDAP URL retrieves the `cn`, and `mail` attributes of the entry for Barbara Jensen.

```
ldap://ldap.example.com/cn=Barbara%20Jensen,dc=example, dc=com?cn,mail
```

- Because no search scope is specified, the search is restricted to the base entry `cn=Barbara Jensen,dc=example,dc=com`.
- Because no filter is specified, the directory uses the default filter `objectclass=*`.

EXAMPLE 12–4 Retrieving the Surname `Jensen` Under `dc=example,dc=com`

The following LDAP URL specifies a search for entries that have the surname `Jensen` and are at any level under `dc=example,dc=com`:

```
ldap://ldap.example.com/dc=example,dc=com??sub?(sn=Jensen)
```

- Because no attributes are specified, the search returns all attributes.
- Because the search scope is `sub`, the search encompasses the base entry `dc=example,dc com` and entries at all levels under the base entry.

EXAMPLE 12–5 Retrieving the Object Class for all Entries One Level Under `dc=example,dc=com`

The following LDAP URL specifies a search for the object class for all entries one level under `dc=example,dc=com`:

```
ldap://ldap.example.com/dc=example,dc=com?objectClass?one
```

- Because the search scope is `one`, the search encompasses all entries one level under the base entry `dc=example,dc=com`. The search scope does not include the base entry.
- Because no filter is specified, the directory uses the default filter `objectclass=*`.

CHAPTER 13

Directory Server LDIF and Search Filters

Directory Server uses the LDAP Data Interchange Format (LDIF) to describe a directory and its entries in text format. LDIF can be used to build the initial directory database or to add large numbers of entries to a directory. LDIF can also be used to describe changes to directory entries. Most command-line utilities rely on LDIF for input or output.

All directory data is stored by using the UTF-8 encoding of Unicode, and, therefore, LDIF files must also be UTF-8 encoded.

This chapter also provides information about searching the directory, and LDAP search filters.

For information about LDIF and searching the directory, see the following sections:

LDIF File Format

LDIF files consist of one or more directory entries separated by a blank line. Each LDIF entry consists of the following parts:

- Entry ID (optional)
- Distinguished name (required)
- One or more object classes
- Multiple attribute definitions

The LDIF format is defined in RFC 2849.

The following example shows a basic directory entry in LDIF.

EXAMPLE 13–1 A Directory Entry in LDIF

```
dn: distinguished_name
objectClass: object_class
objectClass: object_class
...
attribute_type[;subtype]: attribute_value
attribute_type[;subtype]: attribute_value
...
```

An LDIF file must contain the following parts:

- A DN
- At least one object class definition
- Any attributes required by the object classes that you define for the entry

All other attributes and object classes are optional. Object classes and attributes can be specified in any order. The space after the colon is optional.

The following table describes the fields in a LDIF file.

TABLE 13–1 LDIF Fields

Field	Definition
[*id*]	Optional. A positive decimal number representing the entry ID. The database creation tools generate this ID for you. Never add or edit this value yourself.
`dn:` *distinguished_name*	The distinguished name for the entry.
`objectClass:` *object_class*	An object class to use with this entry. The object class identifies the types of attributes or schema that are allowed and required for the entry.
attribute_type	A descriptive attribute to use with the entry. The attribute should be defined in the schema.
[*subtype*]	Optional. A subtype of one of the following types: ▪ Language (*attribute*`;lang-`*subtype*) identifies the language in which the corresponding attribute value is expressed ▪ Binary (*attribute*`;binary`) identifies whether the attribute value is binary ▪ Pronunciation (*attribute*`;phonetic`) identifies whether the attribute value is a pronunciation of an attribute value
attribute_value	The attribute value to be used with the attribute type.

The LDIF syntax for representing a change to an entry in the directory is different from the syntax described above.

Continuing Lines in LDIF

When you specify LDIF, you can break and continue a line or fold a line by indenting the continued portion of the line by one space. For example, the following two statements are identical:

```
dn: cn=Babs Jensen,dc=example,dc=com

dn: cn=Babs J
 ensen,dc=exam
 ple,dc=com
```

You are not required to break and continue LDIF lines. However, doing so can improve the readability of an LDIF file.

Binary Data in LDIF

You can represent binary data in LDIF by using one of the following methods:

- Standard LDIF notation, the lesser than, <, symbol
- Command-line utility, `ldapmodify` with the `-b` option
- Base 64 encoding

Representing Binary Data by Using Standard LDIF Notation

The following example gives the standard LDIF notation of binary data:

```
jpegphoto:< file:/path/to/photo
```

In the example, the path is relative to the client, not to the server. To use standard notation, you do not need to specify the `ldapmodify -b` parameter. However, you must add the following line to the beginning of your LDIF file or to your LDIF update statements:

```
version:1
```

For example, you could use the `ldapmodify` command, as follows:

```
$ ldapmodify -D userDN -w passwd
version: 1
dn: cn=Barbara Jensen,ou=People,dc=example,dc=com
changetype: modify
add: userCertificate
userCertificate;binary:< file: BabsCert
```

Representing Binary Data by Using the `ldapmodify -b` Command

For backward compatibility with earlier versions of Directory Server, binary data can be represented by using the `ldapmodify -b` command. However, when possible, use the standard LDIF notation to represent binary data.

Directory Server accepts the `ldapmodify` command with the `-b` parameter and the following LDIF notation:

```
jpegphoto: /path/to/photo
```

This notation indicates that the `ldapmodify` command should read the referenced file for binary values if the attribute value begins with a slash.

Representing Binary Data by Using Base 64 Encoding

Base 64 encoded data is represented by the `::` symbol, as shown in this example:

```
jpegPhoto:: encoded_data
```

In addition to binary data, the following values must be base 64 encoded:

- Any value that begins with a semicolon, `;`, or a space
- Any value that contains non ASCII data, including new lines

Use the `ldif` command with the `-b` parameter to convert binary data to LDIF format, as follows.

```
$ ldif -b attributeName
```

For more information about how to use the `ldif` command, see the `ldif`(1) man page.

In the above example, *attributeName* is the name of the attribute to which you are supplying the binary data. The binary data is read from standard input and the results are written to standard output. Use redirection operators to select input and output files.

The command takes any input and formats it with the correct line continuation and appropriate attribute information. The command also assesses whether the input requires base–64 encoding. The following example takes a binary file containing a JPEG image and converts it into LDIF format for the attribute named `jpegPhoto`. The output is saved to `out.ldif`:

```
$ ldif -b jpegPhoto < aphoto.jpg > out.ldif
```

The `-b` option specifies that the utility should interpret the entire input as a single binary value. If the `-b` option is not present, each line is considered as a separate input value.

You can edit the output file to add the LDIF statements required to create or modify the directory entry that will contain the binary value. For example, you can open the file `out.ldif` in a text editor and add the following lines at the top of the file.

```
dn: cn=Barbara Jensen,ou=People,dc=example,dc=com
changetype: modify
add: jpegPhoto
jpegPhoto:: encoded_data
```

In this example, *encoded_data* represents the contents of the `out.ldif` file produced by the command.

Directory Entries in LDIF

This section covers the following topics:

- "Organization Entries in LDIF" on page 223
- "Organizational Unit Entries in LDIF" on page 224
- "Organizational Person Entries in LDIF" on page 225

Organization Entries in LDIF

Directories often have at least one organization entry. Typically the organization entry is the first, or topmost entry in the directory. The organization entry often corresponds to the suffix set for the directory. For example, a directory defined to use a suffix of `o=example.com` will probably have an organization entry named `o=example.com`.

The LDIF that defines an organization entry should appear as follows:

```
dn: distinguished_name
objectClass: top
objectClass: organization
o: organization_namelist_of_optional_attributes...
```

The following is an example organization entry in LDIF format:

```
dn: o=example.com
objectclass: top
objectclass: organization
o: example.com Corporation
description: Fictional company for example purposes
telephonenumber: 555-5555
```

The organization name in the following example uses a comma:

```
dn: o=example.com Chile\, S.A.
objectclass: top
objectclass: organization
o: example.com Chile\, S.A.
description: Fictional company for example purposes
telephonenumber: 555-5556
```

The following table describes each element of the organization entry.

TABLE 13–2 Organization Entries in LDIF

LDIF Element	Description
`dn:` *distinguished_name*	Required. Specifies the distinguished name for the entry.
`objectClass: top`	Required. Specifies the top object class.
`objectClass: organization`	Specifies the organization object class. This line defines the entry as an organization.
`o:` *organization_name*	Specifies the organization's name. If the organization name includes a comma, you must escape the comma by a single backslash or the entire organization argument must be enclosed in quotation marks. However, if you are working with a UNIX shell, you must also escape the backslash. Therefore, you must use two back slashes. For example, to set the suffix to `example.com Bolivia, S.A.` you would enter `o: example.com Bolivia\, S.A.`.
list_of_attributes	Specifies the list of optional attributes that you want to maintain for the entry.

Organizational Unit Entries in LDIF

In a directory tree, an organizational unit represents a major subdirectory. A directory tree usually contains more than one organizational unit. An LDIF file that defines an organizational unit entry must appear as follows:

```
dn: distinguished_name
objectClass: top
objectClass: organizationalUnit
ou: organizational_unit_namelist_of_optional_attributes...
```

The following example shows an organizational unit entry in LDIF format:

```
dn: ou=people, o=example.com
objectclass: top
objectclass: organizationalUnit
ou: people
description: Fictional organizational unit for example purposes
```

The following table defines each element of the organizational unit entry.

TABLE 13–3 Organizational Unit Entries in LDIF

LDIF Element	Description
`dn:` *distinguished_name*	Required. Specifies the distinguished name for the entry. If there is a comma in the DN, the comma must be escaped with a backslash (\). For example: `dn: ou=people,o=example.com Bolivia\,S.A.`
`objectClass: top`	Required. Specifies the `top` object class.
`objectClass: organizationalUnit`	Specifies the `organizationalUnit` object class. This line defines the entry as an `organizationalUnit.`
`ou:` *organizational_unit_name*	Specifies an attribute containing the name of the organizational unit.
list_of_attributes	Specifies the list of optional attributes that maintain for the entry.

Organizational Person Entries in LDIF

The majority of the entries in a directory represent organizational people. In LDIF, the definition of an organizational person is as follows:

```
dn: distinguished_name
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: common_name
sn: surname
list_of_optional_attributes
```

The following example shows an organizational person entry in LDIF format:

```
dn: uid=bjensen,ou=people,o=example.com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: Babs Jensen
sn: Jensen
givenname: Babs
uid: bjensen
ou: Marketing
ou: people
description: Fictional person for example purposes
```

```
telephonenumber: 555-5557
userpassword: {sha}dkfljlk34r2kljdsfk9
```

The following table defines each element of the LDIF person entry.

TABLE 13–4 Organizational Person Entries in LDIF

LDIF Element	Description
`dn:` *distinguished_name*	Required. Specifies the distinguished name for the entry. If there is a comma in the DN, the comma must be escaped with a backslash (\). For example, `dn:uid=bjensen,ou=people,o=example.com Bolivia\,S.A.`
`objectClass: top`	Required. Specifies the `top` object class.
`objectClass: person`	Specifies the `person` object class. This object class specification should be included because many LDAP clients require it during search operations for a person or an organizational person.
`objectClass: organizationalPerson`	Specifies the `organizationalPerson` object class. This object class specification should be included because some LDAP clients require it during search operations for an organizational person.
`objectClass: inetOrgPerson`	Specifies the `inetOrgPerson` object class. The `inetOrgPerson` object class is recommended for the creation of an organizational person entry because this object class includes the widest range of attributes. The `uid` attribute is required by this object class, and entries that contain this object class are named based on the value of the `uid` attribute.
`cn:` *common_name*	Required. Specifies the person's common name which is the full name commonly used by the person. For example, `cn: Bill Anderson`.
`sn:` *surname*	Required. Specifies the person's surname, or last name. For example, `sn: Anderson`.
list_of_attributes	Specifies the list of optional attributes that you maintain for the entry.

Guidelines for Defining Directories by Using LDIF

Follow these guidelines to create a directory by using LDIF.

- Create an ASCII file that contains the entries you want to add in LDIF format.
- Separate entries with a single empty line. Do not allow the first line of the file to be blank, otherwise the `ldapmodify` command will exit.
- Begin each file with the topmost, or root, entry in the database. The root entry must represent the suffix or sub-suffix contained by the database. For example, if your database has the suffix `dc=example,dc=com`, the first entry in the directory must be

  ```
  dn: dc=example,dc=com
  ```
- Create the branch point for a subtree before you create entries to go in the subtree.
- Create the directory from the LDIF file by using one of the following methods:
  - By using the Directory Service Control Center
  - By using the `dsadm` command and `dsconf` command
  - By using the`ldapmodify` command with the `-a` option or `-B` option

  Create the directory by using `ldapmodify` command if you currently have a directory database but you are adding a new subtree to the database. Unlike the other methods for creating the directory from an LDIF file, Directory Server must be running before you can add a subtree by using the `ldapmodify` command.

The following example shows an LDIF file with one organization entry, two organizational unit entries, and three organizational person entries.

EXAMPLE 13–2 Example LDIF File With Entries for Organization, Organizational Units, and Organizational Person

```
dn: o=example.com Corp
objectclass: top
objectclass: organization
o: example.com Corp
description: Fictional organization for example purposes
dn: ou=People,o=example.com Corp
objectclass: top
objectclass: organizationalUnit
ou: People
description: Fictional organizational unit for example purposes
tel: 555-5559

dn: cn=June Rossi,ou=People,o=example.com Corp
objectClass: top
objectClass: person
objectClass: organizationalPerson
```

EXAMPLE 13–2 Example LDIF File With Entries for Organization, Organizational Units, and Organizational Person *(Continued)*

```
objectClass: inetOrgPerson
cn: June Rossi
sn: Rossi
givenName: June
mail: rossi@example.com
userPassword: {sha}KDIE3AL9DK
ou: Accounting
ou: people
telephoneNumber: 2616
roomNumber: 220

dn: cn=Marc Chambers,ou=People,o=example.com Corp
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Marc Chambers
sn: Chambers
givenName: Marc
mail: chambers@example.com
userPassword: {sha}jdl2alem87dlacz1
telephoneNumber: 2652
ou: Manufacturing
ou: People
roomNumber: 167

dn: cn=Robert Wong,ou=People,o=example.com Corp
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Robert Wong
cn: Bob Wong
sn: Wong
givenName: Robert
givenName: Bob
mail: bwong@example.com
userPassword: {sha}nn2msx761
telephoneNumber: 2881
roomNumber: 211
ou: Manufacturing
ou: people

dn: ou=Groups,o=example.com Corp
objectclass: top
```

EXAMPLE 13–2 Example LDIF File With Entries for Organization, Organizational Units, and Organizational Person *(Continued)*

```
objectclass: organizationalUnit
ou: groups
description: Fictional organizational unit for example purposes
```

Storing Information in Multiple Languages

For directories that contains a single language, it is not necessary to do anything special to add a new entry to the directory. However, for multinational organizations, it can be necessary to store information in multiple languages so that users in different locales can view directory information in their own language.

When information is represented in multiple languages, the server associates language tags with attribute values. When a new entry is added, attribute values used in the RDN (Relative Distinguished Name) must be added without any language codes.

Multiple languages can be stored within a single attribute. The attribute type is the same, but each attribute value has a different language code. The language tag has no effect on how the string is stored within the directory. All object class and attribute strings are stored using UTF-8.

For a list of the languages supported by Directory Server and their associated language tags, refer to “Identifying Supported Locales” on page 202.

For example, the example.com Corporation has offices in the United States and France. The company wants employees to be able to view directory information in their native language. When a directory entry is added for a new employee, Babs Jensen, the administrator creates the entry in LDIF. The administrator creates values for the `personalTitle` attribute in English and French, as follows:

```
dn: uid=bjensen,ou=people, o=example.com Corp
objectclass: top
objectclass: person
objectclass: organizationalPerson
name: Babs Jensen
cn: Babs Jensen
sn: Jensen
uid: bjensen
personalTitle: Miss
personalTitle;lang-en: Miss
personalTitle;lang-fr: Mlle
preferredLanguage: fr
```

Users accessing this directory entry with an LDAP client with the preferred language set to English will see the personal title `Miss`. Users accessing the directory with an LDAP client with the preferred language set to French will see the title `Mlle`.

Guidelines for Providing LDIF Input

All directory data is stored using the UTF-8 encoding of Unicode. Therefore, any LDIF input you provide must also be UTF-8 encoded. The LDIF format is described in detail in “LDAP Data Interchange Format Reference” in the *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

Consider the following points when you provide LDIF input:

- An object is a blank line followed by a line that starts with `dn:`. This line is the distinguished name of the object. All other lines are the object’s attributes.
- Comments start with # (and end with the EOL.)
- Lines starting with a single space continue the previous line.
- Binary values are base-64 encoded, and represented with a double colon (`::`) after the attribute name.
- Carriage returns and line feeds unsafe in LDIF values, and should be base-64 encoded.
- Do not unintentionally leave trailing spaces at the end of an attribute value when you change the attribute value by using the `ldapmodify` command.

Terminating LDIF Input on the Command Line

The `ldapmodify` and `ldapdelete` utilities read the LDIF statements that you enter after the command in exactly the same way as if they were read from a file. When you finish providing input, enter the character that your shell recognizes as the end of file (EOF) escape sequence.

Typically, the EOF escape sequence is Control-D (`^D`).

The following example shows how to terminate input to the `ldapmodify` command:

```
prompt\> ldapmodify -h host -p port -D cn=admin,cn=Administrators,cn=config -w -
 dn: cn=Barry Nixon,ou=People,dc=example,dc=com
changetype: modify
delete: telephonenumber
^D
prompt\>
```

For simplicity and portability, examples in this document do not show prompts or EOF sequences.

Using Special Characters

When entering command options on the command line, you may need to escape characters that have special meaning to the command-line interpreter, such as space ( ), asterisk (*), backslash (\\), and so forth. For example, many DNs contain spaces, and you must enclose the value in double quotation marks ("") for most UNIX shells:

```
-D "cn=Barbara Jensen,ou=Product Development,dc=example,dc=com"
```

Depending on your command-line interpreter, you should use either single or double quotation marks for this purpose. Refer to your operating system documentation for more information.

In addition, if you are using DNs that contain commas, you must escape the commas with a backslash (\\). For example:

```
-D "cn=Patricia Fuentes,ou=People,o=example.com Bolivia\\,S.A."
```

Note that LDIF statements after the `ldapmodify` command are being interpreted by the command, not by the shell, and therefore do not need special consideration.

Using Attribute OIDs

Attribute OIDs are by default not supported in attribute names. This was not the case in some previous versions of Directory Server. If you used attribute OIDs as attribute names in a previous version of Directory Server, you must set the attribute `nsslapd-attribute-name-exceptions` to `on` for the attribute OIDs to be accepted.

Schema Checking

When adding or modifying an entry, the attributes you use must be required or allowed by the object classes in your entry, and your attributes must contain values that match their defined syntax.

When modifying an entry, Directory Server performs schema checking on the entire entry, not only the attributes being modified. Therefore, the operation may fail if any object class or attribute in the entry does not conform to the schema.

Ordering of LDIF Entries

In any sequence of LDIF text for adding entries, either on the command line or in a file, parent entries must be listed before their children. This way, when the server process the LDIF text, it will create the parent entries before the children entries.

For example, if you want to create entries in a People subtree that does not exist in your directory, then list an entry representing the People container before the entries within the subtree:

```
dn: dc=example,dc=com
dn: ou=People,dc=example,dc=com
...
People subtree entries...
dn: ou=Group,dc=example,dc=com
...
Group subtree entries...
```

You can use the `ldapmodify` command-line utility to create any entry in the directory, however, the root of a suffix or subsuffix is a special entry that must be associated with the necessary configuration entries.

Managing Large Entries

Before adding or modifying entries with very large attribute values, you may need to configure the server to accept them. To protect against overloading the server, clients are limited to sending data no larger than 2 MB by default.

If you add an entry larger than this, or modify an attribute to a value which is larger, the server will refuse to perform the operation and immediately close the connection. For example, binary data such as multi-media contents in one or more attributes of an entry may exceed this limit.

Also, the entry defining a large static group may contain so many members that their representation exceeds the limit. However, such groups are not recommended for performance reasons, and you should consider redesigning your directory structure.

▼ To Modify the Size Limit Enforced by the Server on Data Sent by Clients

1 **Set a new value for the `nsslapd-maxbersize` attribute of the `cn=config` entry.**

- To do this from the command line, use the following command:

  ```
  ldapmodify -h host -p port -D cn=admin,cn=Administrators,cn=config -w -
  dn: cn=config
  changetype: modify
  replace: nsslapd-maxbersize
  nsslapd-maxbersize: sizeLimitInBytes
   ^D
  ```

  For more information, see “`nsslapd-maxbersize`” in the *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

2 Restart the server.

Error Handling

The command-line tools process all entries or modifications in the LDIF input sequentially. The default behavior is to stop processing when the first error occurs. Use the `-c` option to continue processing all input regardless of any errors. You will see the error condition in the output of the tool.

In addition to the considerations listed above, common errors are:

- Not having the appropriate access permission for the operation.
- Adding an entry with a DN that already exists in the directory.
- Adding an entry below a parent that does not exist.

Searching the Directory

You can locate entries in a directory using any LDAP client. Most clients provide some form of search interface that enables you to search the directory and retrieve entry information.

The access control that has been set in your directory determines the results of your searches. Common users typically do not "see" much of the directory, and directory administrators have full access to all data, including configuration.

Searching the Directory With ldapsearch

You can use the `ldapsearch` command-line utility to locate and retrieve directory entries. Note that the `ldapsearch` utility described in this section is not the utility provided with the Solaris platform, but is part of the Directory Server Resource Kit.

This utility opens a connection to the server with a specified a user identity (usually a distinguished name) and password, and locates entries based on a search filter. Search scopes can include a single entry, an entry's immediate subentries, or an entire tree or subtree.

Search results are returned in LDIF format.

ldapsearch Command-Line Format

When you use `ldapsearch`, you must enter the command using the following format:

`ldapsearch [`*optional_options*`] [`*search_filter*`] [`*optional_list_of_attributes*`]`

where

- *optional_options* represents a series of command-line options. These must be specified before the search filter, if any.
- *search_filter* represents an LDAP search filter in a file using the `-f` option.
- *optional_list_of_attributes* represents a list of attributes separated by a space. Specifying a list of attributes reduces the number of attributes returned in the search results. This list of attributes must appear after the search filter. If you do not specify a list of attributes, the search returns values for all attributes permitted by the access control set in the directory (with the exception of operational attributes).

Note – If you want operational attributes returned as a result of a search operation, you must explicitly specify them in the search command. To retrieve regular attributes in addition to explicitly specified operational attributes, use an asterisk (*) in the list of attributes in the ldapsearch command.

Using Special Characters

When using the `ldapsearch` command-line utility, you may need to specify values that contain characters that have special meaning to the command-line interpreter (such as space [ ], asterisk [*], backslash [\\], and so forth). When you specify special characters, enclose the value in quotation marks (“”). For example:

```
-D "cn=Charlene Daniels,ou=People,dc=example,dc=com"
```

Depending on your command-line interpreter, use either single or double quotation marks for this purpose. Refer to your shell documentation for more information.

Commonly Used ldapsearch options

The following lists the most commonly used `ldapsearch` command-line options. If you specify a value that contains a space [ ], the value should be surrounded by double quotation marks, for example, `-b "ou=groups, dc=example,dc=com"`.

`-b` Specifies the starting point for the search. The value specified here must be a distinguished name that currently exists in the database. This option is optional if the `LDAP_BASEDN` environment variable has been set to a base DN.

The value specified in this option should be provided in double quotation marks. For example:

```
-b "cn=Charlene Daniels, ou=People, dc=example,dc=com"
```

`-D` Specifies the distinguished name with which to authenticate to the server. This option is optional if anonymous access is supported by your server. If specified, this value must be a DN recognized by Directory Server, and it must also have the authority to search for the entries. For example:

-D "uid=cdaniels, dc=example,dc=com"

-h Specifies the hostname or IP address of the machine on which Directory Server is installed. If you do not specify a host, ldapsearch uses the localhost. For example, -h myServer.

-l Specifies the maximum number of seconds to wait for a search request to complete. Regardless of the value specified here, ldapsearch will never wait longer than is allowed by the server's nsslapd-timelimit attribute (except in the case of a persistent search.)*Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

For example, -l 300. The default value for the nsslapd-timelimit attribute is 3,600 seconds (1 hour.)

-p Specifies the TCP port number that Directory Server uses. For example, -p 5201. The default is 389, and 636 when the SSL options are used.

-s Specifies the scope of the search. The scope can be one of:

- base—Search only the entry specified in the -b option or defined by the LDAP_BASEDN environment variable.
- one—Search only the immediate children of the entry specified in the -b option. Only the children are searched; the actual entry specified in the -b option is not searched.
- sub—Search the entry specified in the -b option and all of its descendants. That is, perform a subtree search starting at the point identified in the -b option. This is the default.

-w Specifies the password associated with the distinguished name that is specified in the -D option. If you do not specify this option, anonymous access is used. For example, -w diner892.

-x Specifies that the search results are sorted on the server rather than on the client. This is useful if you want to sort according to a matching rule, as with an international search. In general, it is faster to sort on the server rather than on the client, although server-side sorting uses server resources.

-z Specifies the maximum number of entries to return in response to a search request. For example, -z 1000.

Normally, regardless of the value specified here, ldapsearch never returns more entries than the number allowed by the server's nsslapd-sizelimit attribute. However, you can override this limitation by binding as the root DN when using this command-line argument. When you bind as the root DN, this option defaults to zero (0). The default value for the nsslapd-sizelimit attribute is 2,000 entries.

For detailed information on all ldapsearch utility options, refer to ldapmodify(1).

ldapsearch Examples

In the next set of examples, the following assumptions are made:

- You want to perform a search of all entries in the directory.
- The server is located on hostname **mySerer**.
- The server uses port number **5201**.
- You are binding to the directory as **cn=admin,cn=Administrators,cn=config**. Using the symbol "-" means that you will be prompted for the password on the command line.
- SSL is enabled for the server on port **636** (the default SSL port number).
- The suffix under which all data is stored is **dc=example,dc=com**.

Returning All Entries

Given the previous information, the following call will return all entries in the directory:

```
ldapsearch -h myServer -p 5201 -D cn=admin,cn=Administrators,cn=config
 -b "dc=example,dc=com" -s sub "(objectclass=*)"
```

"(objectclass=*)" is a search filter that matches any entry in the directory.

Specifying Search Filters on the Command Line

You can specify a search filter directly on the command line. If you do this, be sure to enclose your filter in quotation marks ("filter"). Also, do not specify the -f option.

For example:

```
ldapsearch -h myServer -p 5201 -D cn=admin,cn=Administrators,cn=config -w -
 -b "dc=example,dc=com" "(cn=Charlene Daniels)"
```

Searching the Root DSE Entry

The root DSE is a special entry that contains information related to the current server instance, such as a list of supported suffixes, available authentication mechanisms, and so forth. You can search this entry by supplying a search base of "". You must also specify a search scope of base and a filter of "(objectclass=*)".

For example:

```
ldapsearch -h myServer -p 5201 -D cn=admin,cn=Administrators,cn=config -w -
 -b "" -s base "(objectclass=*)"
```

Searching the Schema Entry

Directory Server stores all directory server schema in the special cn=schema entry. This entry contains information on every object class and attribute defined for your directory server.

You can examine the contents of this entry as follows:

```
ldapsearch -h myServer -p 5201 -D cn=admin,cn=Administrators,cn=config
 -b "cn=schema" -s base "(objectclass=*)"
```

Note – For strict compliance, the location of the schema subentry for a given entry is specified by the subschemaSubentry operational attribute. In this version of Directory Server, the value of this attribute is always cn=schema.

Using LDAP_BASEDN

To make searching easier, you can set your search base using the LDAP_BASEDN environment variable. Doing this allows you to skip specifying the search base with the -b option (for information on how to set environment variables, see the documentation for your operating system).

Typically, you set LDAP_BASEDN to your directory's suffix value. Since your directory suffix is equal to the root, or topmost, entry in your directory, this causes all searches to begin from your directory's root entry.

For example, if you have set LDAP_BASEDN to dc=example,dc=com, you can search for (cn=Charlene Daniels) in your directory using the following command-line call:

```
ldapsearch -h myServer -p 5201 -D cn=admin,cn=Administrators,cn=config -w -
 "(cn=Charlene Daniels)"
```

In this example, the default scope of sub is used because the -s option was not used to specify the scope.

Displaying Subsets of Attributes

The ldapsearch command returns all search results in LDIF format. By default, ldapsearch returns the entry's distinguished name and all of the attributes that you are allowed to read. You can set up the directory access control such that you are allowed to read only a subset of the attributes on any given directory entry.) Only operational attributes are not returned. If you want operational attributes returned as a result of a search operation, you must explicitly specify them in the search command. For more information on operational attributes, refer to the TODO: No more AdminServerAdminGuide.

Suppose you do not want to see all of the attributes returned in the search results. You can limit the returned attributes to just a few specific attributes by specifying the ones you want on the command line immediately after the search filter. For example, to show the cn and sn attributes for every entry in the directory, use the following command:

```
ldapsearch -h myServer -p 5201 -D cn=admin,cn=Administrators,cn=config -w -
 "(objectclass=*)" sn cn
```

This example assumes you set your search base with LDAP_BASEDN.

Searching Multi-Valued Attributes

During a search, Directory Server does not necessarily return multi-valued attributes in sorted order. For example, suppose you want to search for configuration attributes on cn=config requiring that the server be restarted before changes take effect.

```
ldapsearch -h myServer -p 5201 -D cn=admin,cn=Administrators,cn=config -w -
 -b cn=config "(objectclass=*)" nsslapd-requiresrestart
```

The following result is returned:

```
dn: cn=config
nsslapd-requiresrestart: cn=config:nsslapd-port
nsslapd-requiresrestart: cn=config:nsslapd-secureport
nsslapd-requiresrestart: cn=config:nsslapd-plugin
nsslapd-requiresrestart: cn=config:nsslapd-changelogdir
nsslapd-requiresrestart: cn=config:nsslapd-changelogsuffix
nsslapd-requiresrestart: cn=config:nsslapd-changelogmaxentries
nsslapd-requiresrestart: cn=config:nsslapd-changelogmaxage
nsslapd-requiresrestart: cn=config:nsslapd-db-locks
nsslapd-requiresrestart: cn=config:nsslapd-return-exact-case
nsslapd-requiresrestart: cn=config,cn=ldbm database,cn=plugins,
  cn=config:nsslapd-allidsthreshold
nsslapd-requiresrestart: cn=config,cn=ldbm database,cn=plugins,
  cn=config:nsslapd-dbcachesize
nsslapd-requiresrestart: cn=config,cn=ldbm database,cn=plugins,
  cn=config:nsslapd-dbncache
nsslapd-requiresrestart: cn=config,cn=ldbm database,cn=plugins,
  cn=config:nsslapd-directory
nsslapd-requiresrestart: cn=encryption,cn=config:nssslsessiontimeout
nsslapd-requiresrestart: cn=encryption,cn=config:nssslclientauth
nsslapd-requiresrestart: cn=encryption,cn=config:nssslserverauth
nsslapd-requiresrestart: cn=encryption,cn=config:nsssl2
nsslapd-requiresrestart: cn=encryption,cn=config:nsssl3
...
```

As shown here, the `nsslapd-requiresrestart` attribute takes multiple values. These values are not, however, in sorted order. If you develop an application that requires multi-valued attributes in sorted order, make sure that your application performs the sort.

Using Client Authentication When Searching

This example shows user `cdaniels` searching the directory using client authentication:

```
ldapsearch -h myServer -p 636 -b "dc=example,dc=com"
  -N  "cdanielsscertname" -Z -W certdbpassword
  -P /home/cdaniels/certdb/cert.db "(givenname=Richard)"
```

LDAP Search Filters

Search filters select the entries to be returned for a search operation. They are most commonly used with the `ldapsearch` command-line utility. When you use `ldapsearch`, you can place multiple search filters in a file, with each filter on a separate line in the file, or you can specify a search filter directly on the command line.

For example, the following filter specifies a search for the common name Lucie Du Bois:

```
(cn=Lucie Du Bois)
```

This search filter returns all entries that contain the common name Lucie Du Bois. Searches for common name values are not case sensitive.

When the common name attribute has values associated with a language tag, all of the values are returned. Thus, the following two attribute values both match this filter:

```
cn: Lucie Du Bois
cn;lang-fr: Lucie Du Bois
```

Search Filter Syntax

The basic syntax of a search filter is:

```
(attribute  operator  value)
```

For example:

```
(buildingname\>=alpha)
```

In this example, `buildingname` is the attribute, `\>=` is the operator, and **`alpha`** is the value. You can also define filters that use different attributes combined together with Boolean operators.

Using Attributes in Search Filters

When searching for an entry, you can specify attributes associated with that type of entry. For example, when you search for people entries, you can use the `cn` attribute to search for people with a specific common name.

Examples of attributes that people entries might include:

- `cn` (the person's common name)
- `sn` (the person's surname, or last name, or family name)
- `telephoneNumber` (the person's telephone number)
- `buildingName` (the name of the building in which the person resides)
- `l` (the locality in which you can find the person)

Using Operators in Search Filters

The operators that you can use in search filters are listed in Table 13–5:

TABLE 13–5 Search Filter Operators

Search Type	Operator	Description
Equality	=	Returns entries containing attribute values that exactly match the specified value. For example, `cn=Bob Johnson`
Substring	=*string***string*	Returns entries containing attributes containing the specified substring. For example, `cn=Bob*cn=*Johnsoncn=*John*cn=B*John` (The asterisk (*) indicates zero (0) or more characters.)
Greater than or equal to	\>=	Returns entries containing attributes that are greater than or equal to the specified value. For example, `buildingname \>= alpha`
Less than or equal to	<=	Returns entries containing attributes that are less than or equal to the specified value. For example, `buildingname <= alpha`

Presence	=*	Returns entries containing one or more values for the specified attribute. For example,
		`cn=*`
		`telephonenumber=*`
		`manager=*`

TABLE 13–5 Search Filter Operators *(Continued)*

Search Type	Operator	Description

Approximate	~=	Returns entries containing the specified attribute with a value that is approximately equal to the value specified in the search filter. For example,
		`cn~=suret`
		`l~=san fransico`
		could return
		`cn=sarette`
		`l=san francisco`
		The Approximate operator is experimental and works only with English language strings. It does not work with non-ASCII based strings, such as Ja or Zn.

Extended operators exist that extend searches to `dn` attributes (`cn:dn:=John`, for example) and provide support for internationalized searches.

Using OIDs in Search Filters

LDAPv3 enables you to build match operators and rules for a particular attribute. Matching rules define how to compare attribute values with a particular syntax. In other words, a matching rule defines how potentially matching attributes are compared. For example, a matching rule can define whether or not to take text case into account when comparing attributes.

When the rules are created, they can be referred to in a search filter.

For example, the following search filter compares entries containing the surname "Jensen" by using the matching rule designated by OID `2.5.13.5`:

```
(sn:2.5.13.5:=Jensen)
```

The following example illustrates the use of the ":dn" notation to indicate that OID `2.5.13.5` should be used when making comparisons, and that the attributes of an entry\qs distinguished name should be considered part of the entry when evaluating the match:

```
(sn:dn:2.5.13.5:=Jensen)
```

Using Compound Search Filters

Multiple search filter components can be combined using Boolean operators expressed in prefix notation as follows:

(*Boolean-operator*(*filter*)(*filter*)(*filter*)...)

where *Boolean-operator* is any one of the Boolean operators listed in Table 13–6.

Boolean operators can be combined and nested together to form complex expressions, such as:

(*Boolean-operator*(*filter*)(*Boolean-operator*(*filter*)(*filter*)))

The Boolean operators available for use with search filters include the following:

TABLE 13–6 Search Filter Boolean Operators

Operator	Symbol	Description
AND	&	All specified filters must be true for the statement to be true. For example, `(&(filter)(filter)(filter)...)`
OR	|	At least one specified filter must be true for the statement to be true. For example, `(|(filter)(filter)(filter)...)`
NOT	!	The specified statement must not be true for the statement to be true. Only one filter is affected by the NOT operator. For example, `(!(filter))` The use of the NOT operator results in an unindexed search.

Boolean expressions are evaluated in the following order:

- Innermost to outermost parenthetical expressions first
- All expressions from left to right

Specifying Search Filters Using a File

You can enter search filters into a file instead of entering them on the command line. When you do this, specify each search filter on a separate line in the file. The `ldapsearch` command runs each search in the order in which it appears in the file.

For example, if the file contains:

```
(sn=Daniels)
(givenname=Charlene)
```

then `ldapsearch` first finds all the entries with the surname Daniels, and then all the entries with the given name Charlene. If an entry is found that matches both search criteria, the entry is returned twice.

For example, suppose you specified the previous search filters in a file named `searchdb`, and you set your search base using `LDAP_BASEDN`. The following returns all the entries that match either search filter:

```
ldapsearch -h myServer -p 5201 -D cn=admin,cn=Administrators,cn=config -w -
 -f searchdb
```

You can limit the set of attributes returned here by specifying the attribute names that you want at the end of the search line. For example, the following `ldapsearch` command performs both searches, but returns only the DN and the `givenname` and `sn` attributes of each entry:

```
ldapsearch -h myServer -p 5201 -D cn=admin,cn=Administrators,cn=config -w -
 -f searchdb sn givenname
```

Specifying Non 7-Bit ASCII Characters in Search Filters

Non 7-bit ASCII characters in search filters must be replaced with a representation of the character, where each byte of the UTF-8 encoding is preceded by a backslash. In UTF-8, characters are represented by a hexadecimal code for each byte.

For example, the character é has UTF-8 representation `c3a9`. Thus, in a search filter, you represent é as `\\c3\\a9`. So, to search for cn=Véronique Martin:

ldapsearch -h myServer -b "dc=example,dc=com" "(cn=V\\c3\\a9ronique Martin)"

The special characters listed in Table 13–7 must also be represented in this fashion when used in search filters.

TABLE 13–7 Special Characters in Search Filters

Special character	Value With Special Character	Example Filter
*	`Five*Star`	`(cn=Five\\2aStar)`
\\	`c:\\File`	`(cn=\\5cFile)`
()	`John (2nd)`	`(cn=John \\282nd\\29)`
null	`0004`	`(bin=\\00\\00\\00\\04)`

Escaped Characters in Distinguished Names within Search Filters

When using a DN in any part of Directory Server, you must escape commas and certain other special characters with a backslash (\\). If you are using a DN in a search filter, the backslash used for escaping special characters in DNs must be represented by `\\5c`. For example:

DN: `cn=Julie Fulmer,ou=Marketing\\,Bolivia,dc=example,dc=com`

DN in a search filter: `ldapsearch -h myServer -b "dc=example,dc=com" "(manager=cn=Julie Fulmer,ou=Marketing\\5c,Bolivia,dc=example,dc=com)"`

Search Filter Examples

The following filter searches for entries containing one or more values for the manager attribute. This is also known as a presence search:

```
(manager=*)
```

The following filter searches for entries containing the common name Ray Kultgen. This is also known as an equality search:

```
(cn=Ray Kultgen)
```

The following filter returns all entries that contain a description attribute that contains the substring X.500:

```
(description=*X.500*)
```

The following filter returns all entries whose organizational unit is Marketing and whose description field does not contain the substring X.500:

```
(&(ou=Marketing)(!(description=*X.500*)))
```

The following filter returns all entries whose organizational unit is Marketing and that have Julie Fulmer or Cindy Zwaska as a manager:

```
(&(ou=Marketing)(|(manager=cn=Julie Fulmer,ou=Marketing,
 dc=example,dc=com)(manager=cn=Cindy Zwaska,ou=Marketing,
 dc=example,dc=com)))
```

The following filter returns all entries that do not represent a person:

```
(!(objectClass=person))
```

Note that the previous filter will have a negative performance impact and should be used as part of a complex search. The following filter returns all entries that do not represent a person and whose common name is similar to printer3b:

```
(&(cn~=printer3b)(!(objectClass=person)))
```

Searching for Operational Attributes

If you want operational attributes returned as a result of a search operation, you must explicitly specify them in the search command.

```
ldapsearch -h myServer -p 5201 -D cn=admin,cn=Administrators,cn=config -w -
 "(objectclass=*)" aci
```

To retrieve regular attributes in addition to explicitly specified operational attributes, specify “*” in addition to the operational attributes. For example:

```
ldapsearch -h myServer -p 5201 -D cn=admin,cn=Administrators,cn=config -w -
 "(objectclass=*)" aci *
```

◆ ◆ ◆ CHAPTER 14

Directory Server File Reference

This chapter describes the files found after you install Directory Server, and after you create server instances.

The examples shown in this chapter are for Solaris systems. File extensions and path separators may differ for your operating system. This chapter includes the following sections.

- "Software Layout for Directory Server" on page 247
- "Directory Server Instance Default Layout" on page 251

If you installed software from native packages, you may also use the packaging commands on your system to list the files installed. For example, after installing from native packages on Solaris systems, you can obtain a full list for a particular package using the `pkgchk -v` *package-name* command.

If you installed software from a zip distribution, find lists of installed files in the *install-path*`/dsee6/data/` directory.

Software Layout for Directory Server

This section describes the file layout you find after installing Directory Server from the zip distribution. All files locations are relative to the path where you installed the product. For information on default native package installation locations, see "Default Paths and Command Locations" on page 24.

install-path`/ds6/`

Directory Server files shared by server instances. This directory houses the following files of interest.

install-path`/ds6/bin/dsadm`

Directory Server command for local administration. See `dsadm`(1M).

install-path`/ds6/bin/dsconf`

Directory Server command for configuration over LDAP. See `dsconf`(1M).

install-path/`ds6/bin/dsmig`
Directory Server command for migration to this version of Directory Server. See `dsmig`(1M).

install-path/`ds6/bin/dsrepair`
Directory Server command for repairing replicated directory entries, not intended for use without help for qualified support personnel. See `dsrepair`(1M).

install-path/`ds6/bin/entrycmp`
Directory Server command for comparing directory entries across replica. See `entrycmp`(1).

install-path/`ds6/bin/fildif`
Directory Server command for filtering LDIF content. See `fildif`(1).

install-path/`ds6/bin/insync`
Directory Server command for examining replica synchronization. See `insync`(1).

install-path/`ds6/bin/mmldif`
Directory Server command for combining LDIF content. See `mmldif`(1).

install-path/`ds6/bin/ns-accountstatus`
Directory Server command for examining whether an account is locked. See `ns-accountstatus`(1M).

install-path/`ds6/bin/ns-activate`
Directory Server command for activating a locked account. See `ns-activate`(1M).

install-path/`ds6/bin/ns-inactivate`
Directory Server command for explicitly locking an account. See `ns-inactivate`(1M).

install-path/`ds6/bin/pwdhash`
Directory Server command for displaying the hashed form of a password value. See `pwdhash`(1).

install-path/`ds6/bin/repldisc`
Directory Server command for discovering a replication topology. See `repldisc`(1).

install-path/`ds6/bin/schema_push`
Directory Server command for pushing schema updates to replicas. See `schema_push`(1M).

install-path/`ds6/etc/`
Registration configuration for Directory Server monitoring agents, not intended to be used directly.

install-path/`ds6/examples/`
Sample Directory Server plug-ins.

install-path/`ds6/include/`
Directory Server plug-in header files.

install-path`/ds6/install/`
Directory Server instance installation templates, not intended to be used directly.

install-path`/ds6/ldif/Example.ldif`
Sample Directory Server LDIF content.

install-path`/ds6/ldif/Example-roles.ldif`
Sample Directory Server LDIF content with grouping based on roles.

install-path`/ds6/lib/`
Shared Directory Server libraries, not intended for use directly.

install-path`/ds6/plugins/`
Directory Server plug-in configuration files, not intended to be used directly.

install-path`/ds6/resources/`
Directory Server resource files, not intended to be used directly.

install-path`/ds6/schema/`
Directory Server instance LDAP schema templates, not intended to be used directly.

install-path`/dscc6/`
Directory Service Control Center agent files shared by multiple Directory Server Enterprise Edition component products. This directory houses the following files of interest.

install-path`/dscc6/bin/dsccmon`
Command to monitor servers managed through Directory Service Control Center. See `dsccmon`(1M).

install-path`/dscc6/bin/dsccreg`
Command to manage the Directory Service Control Center registry. See `dsccreg`(1M).

install-path`/dscc6/bin/dsccsetup`
Command to set up Directory Service Control Center. See `dsccsetup`(1M).

install-path`/dscc6/etc/`
Directory Service Control Center agent configuration information, not intended to be used directly.

install-path`/dscc6/lib/`
Shared libraries, not intended to be used directly.

install-path`/dsee6/`
Files shared by multiple Directory Server Enterprise Edition component products. This directory houses the following files of interest.

install-path`/dsee6/bin/certutil`
NSS certificate manipulation command used by other tools, not intended to be used directly.

install-path`/dsee6/bin/dsee_deploy`
Command to install and remove software. See `dsee_deploy`(1M).

install-path/dsee6/bin/ldif
: Command to base64 encode LDIF attribute values. See `ldif`(1)

install-path/dsee6/cacao_2.0/
: Common agent container files shared by Directory Server Enterprise Edition component products, not intended to be used directly.

install-path/dsee6/data/
: Lists of installed files used by the `dsee_deploy` command, not intended to be used directly.

install-path/dsee6/lib/
: Libraries shared by Directory Server Enterprise Edition component products, not intended to be used directly.

install-path/dsee6/man/
: Directory Server Enterprise Edition online reference manual pages. See also *Sun Java System Directory Server Enterprise Edition 6.3 Man Page Reference*.

install-path/dsrk6/bin/ldapcmp
: Directory Server Resource Kit command to compare LDAP entries from two directories. See `ldapcmp`(1).

install-path/dsrk6/bin/ldapcompare
: Directory Server Resource Kit command to perform LDAP compare operations. See `ldapcompare`(1).

install-path/dsrk6/bin/ldapdelete
: Directory Server Resource Kit command to delete directory entries. See `ldapdelete`(1).

install-path/dsrk6/bin/ldapmodify
: Directory Server Resource Kit command to update entries over LDAP. See `ldapmodify`(1).

install-path/dsrk6/bin/ldappasswd
: Directory Server Resource Kit command to change user passwords. See `ldappasswd`(1).

install-path/dsrk6/bin/ldapsearch
: Directory Server Resource Kit command to search a directory. See `ldapsearch`(1).

install-path/dsrk6/lib/
: Libraries used by Directory Server Resource Kit commands, not intended to be used directly.

install-path/etc/
: Container for configuration files.

install-path/jre/
: Java Runtime Environment, not intended to be used directly.

install-path/var/
: Container for runtime files, not intended to be used directly.

Directory Server Instance Default Layout

This section describes the file layout you find after creating a Directory Server instance. The *instance-path* is the file system path where you created the instance.

This section does not cover the following deprecated scripts, generated for backwards compatibility:

- `bak2db`
- `db2bak`
- `db2ldif`
- `ldif2db`
- `restart-slapd`
- `start-slapd`
- `stop-slapd`

If you do not use the scripts, you can safely remove them.

instance-path`/alias/`
NSS certificate database directory.

instance-path`/alias/certmap.conf`
NSS certificate mapping configuration file.

instance-path`/bak/`
Default data backup directory.

Each directory database backup is held in its own file system directory. The name of the backup directory corresponds to the time and date of the backup.

instance-path`/confbak/`
Default configuration backup directory provided for you to backup up versions of your configuration.

instance-path`/conf_bk/`
Default server configuration backup directory.

This directory contains a backup copy of the original Directory Server configuration file, `dse.ldif`, generated when the server instance was created. This copy can be compared with the current configuration file for troubleshooting.

instance-path`/config/`
Server configuration directory.

instance-path`/config/dse.ldif`
Server configuration file, not intended to be edited directly.

instance-path`/config/schema/`
LDAP schema configuration files. See `dirserv`(5dssd).

instance-path/db/

Default server database files directory. When a suffix has been created, the following database files are stored in this file system directory.

`__db.00`*x*	Files used internally by the database. Do not move, delete, or modify these files.
`DBVERSION`	File that identifies the version of the database.
`guardian`	File used to store information about the state of the database, used to determine whether database recovery is required.
`log.`*xxxxxxxxxx*	Files used to store the database transaction logs.
suffix	Files that store your directory suffix information. The directory name is derived from the suffix name, such that the database for a suffix identified by DN `dc=example,dc=com` is stored in a file system directory named `example`. For every index defined in the database, the suffix directory contains a file with a name of the form *suffix_indexedAttr*`.db3`, such that an index of CNs for `dc=example,dc=com` has file name `example_cn.db3`. Suffix directories also contain a file named *suffix*`_id2entry.db3`. The *suffix*`_id2entry.db3` file contains the directory database entries. If necessary, all index files can be rebuild from the *suffix*`_id2entry.db3` file. To recreate the index files, reindex the suffix.

instance-path/locks/

Lock files directory.

Lock files stored here in subdirectories `exports/`, `imports/`, and `server/` prevent simultaneous operations from conflicting with each other. The lock mechanisms allow one server instance to run at a time. The lock mechanisms also permit only one `dsadm import` (offline import) operation at a time. As a result, no export or server instance operations can be run during import.

The lock restriction does not however apply to `dsconf import` (online import) operations. Multiple online imports can run at the same time.

instance-path/logs/

Default server logs directory. The following files are stored here.

`access` logs
This file records information about client access to Directory Server. For detail about `access` logs, see "Access Logs" on page 163.

`audit` logs
This file records information about modifications to Directory Server data. For detail about `audit` logs, see "Audit Logs" on page 164.

`core` files
By default, server `core` files are dumped here during a crash.

`errors` logs
This file records errors, warnings, and informational messages logged during Directory Server operation. For detail about `errors` logs, see "Error Logs" on page 164.

`pid` file
This file holds the process identifier of the running server.

`slapd.stats` file
This memory mapped-file cannot be read in an editor. The `slapd.stats` file contains data collected for SNMP, which is communicated to the agent responsible for handling SNMP requests.

instance-path`/plugins/signatures/`
Plug-in signatures directory, not intended to be used directly.

instance-path`/tmp/`
Server runtime files directory, not intended to be used directly.

PART II

Directory Proxy Server Reference

This part explains how Directory Proxy Server works. The information here is primarily descriptive. For instructions, try Part II, "Directory Proxy Server Administration," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide* instead.

This part includes the following chapters:

- Chapter 15, "Directory Proxy Server Overview," outlines the architecture of Directory Proxy Server and describes, at a high level, the most important features of this release.
- Chapter 16, "Directory Proxy Server Load Balancing and Client Affinity," describes how Directory Proxy Server can be configured for load balancing, and how client affinity can be used to reduce the risk of propagation delay in load balanced deployments.
- Chapter 17, "Directory Proxy Server Distribution," describes how data in an LDAP server is exposed to a client request.
- Chapter 18, "Directory Proxy Server Virtualization," explains how virtual data views enable you to display physical data in a different way, and describes the kinds of virtual data views that are available in Directory Proxy Server.
- Chapter 19, "Connections Between Directory Proxy Server and Backend LDAP Servers," describes the connections between Directory Proxy Server and backend LDAP servers.
- Chapter 20, "Connections Between Clients and Directory Proxy Server," describes how connection handlers are used to apply limits and filters to a connection, and to restrict the data to which clients are exposed.

- Chapter 21, "Directory Proxy Server Client Authentication," describes the client authentication mechanisms available in Directory Proxy Server.
- Chapter 22, "Security in Directory Proxy Server," describes the mechanisms that can be used to secure data that passes through Directory Proxy Server.
- Chapter 23, "Directory Proxy Server Logging," provides an overview of the Directory Proxy Server logging interface.
- Chapter 24, "Directory Proxy Server Alerts and Monitoring," describes the mechanisms that can be used to monitor both Directory Proxy Server and the availability of backend LDAP servers.
- Chapter 25, "Directory Proxy Server File Reference," describes the files found after you install Directory Proxy Server, and after you create server instances.

CHAPTER 15

Directory Proxy Server Overview

This chapter outlines the architecture of Directory Proxy Server, and describes at a high level, the most important features of this release.

The chapter covers the following topics:

- "Introduction to Directory Proxy Server" on page 257
- "Directory Proxy Server Architecture" on page 258
- "Overview of Directory Proxy Server Features" on page 260

Introduction to Directory Proxy Server

Directory Proxy Server is an LDAP application-layer protocol gateway. Directory Proxy Server delivers enhanced directory access control, schema compatibility, and high availability.

The Directory Proxy Server architecture enables you to configure several objects that control how client requests are routed to backend data sources. These configuration objects are illustrated at a high level in the following simplified schematic of the Directory Proxy Server architecture. This illustration will help you to understand the architectural concepts presented in the remainder of this book.

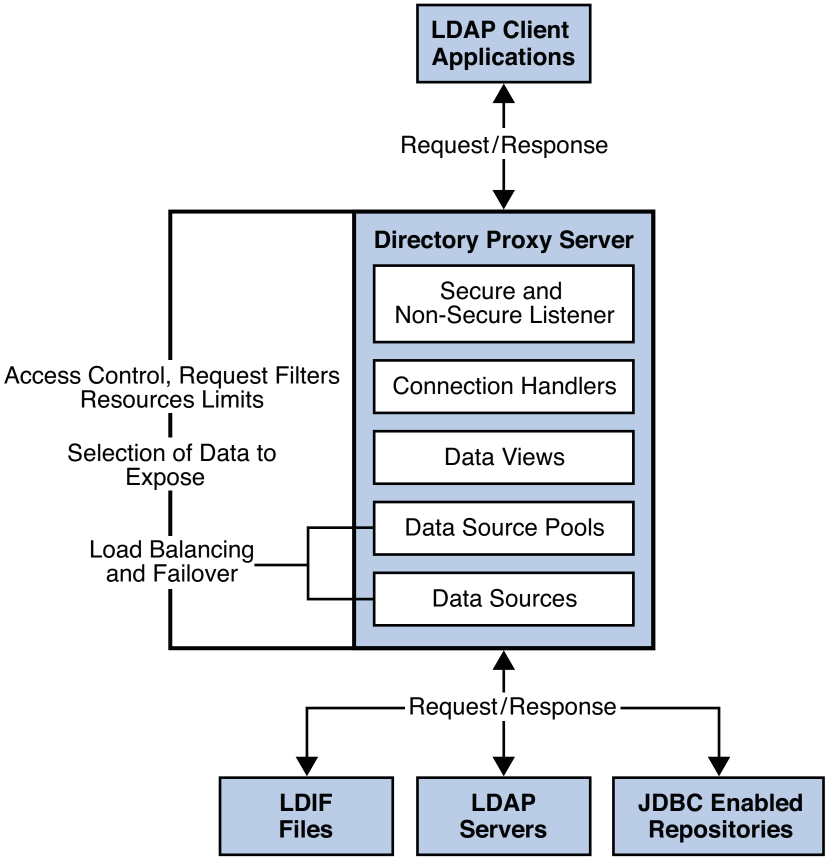


FIGURE 15–1 Simplified Architecture of Directory Proxy Server

Directory Proxy Server Architecture

This section briefly presents the new Directory Proxy Server architecture and what is new compared to 5.x. Its aim is to help you understand why literal translation of some 5.x configuration attributes is not possible.

A Directory Proxy Server instance proxies client application requests to *data sources* through *data views*. Data sources and pools of data sources correspond to load balanced groups from 5.x.

Data views, however, are new. They do not correspond to anything present in 5.x. Fundamentally Directory Proxy Server handles incoming connections individually, assigning a *connection handler* when the connection is opened, and reassigning a connection handler upon rebind when the bind identity changes.

The connection handler gives Directory Proxy Server a set of policy rules for making decisions about what to do with operations requested through a given connection. Connection handlers correspond roughly to network groups in 5.x, yet whereas network groups are configured to use load balanced groups directly.

Directory Proxy Server uses connection handlers mainly to determine policies about a connection, so it can take appropriate decisions about operations performed on that connection. For example, if a connection handler is configured to prevent write operations on a certain connection, Directory Proxy Server can use that property of the policy to short circuit evaluations concerning write operation requests on that connection. In this case, the appropriate errors are returned to the client as soon as Directory Proxy Server has decoded the operation.

LDAP operations on a connection are handled in Directory Proxy Server first through data views. Data views enable Directory Proxy Server to perform DN-based routing. In other words, operations concerning one set of data can be sent to one set of data sources, and operations concerning another set of data can be sent elsewhere. This new architectural form seems unnecessary when you look at it from the point of view of reproducing a 5.x configuration. Yet data views become indispensable when you want to distribute different directory data across various directories, or when you want to recover different data from disparate data sources to present a virtual directory view of those sources to a client application.

Data views therefore enable Directory Proxy Server to select the data sources via a data source pool to handle the LDAP operation. Data source pools, which correspond to 5.x load balanced groups, represent sets of data sources each holding equivalent data. A pool defines the load balancing and failover management that Directory Proxy Server performs to spread load across different data sources. As load balancing is performed per operation, the balancing itself is by nature operation based.

Data sources can be understood as sources of data for reads, and sinks of data for writes. Directory Proxy Server handles the following kinds of data sources:

- LDAP directories
- LDIF files
- JDBC-enabled data repositories

Directory Proxy Server 5.x was essentially a connection based router. In Directory Proxy Server 5.x, a client connection was routed to a directory server. All requests from that client connection were sent to the same directory server until the connection was broken. For compatibility, Directory Proxy Server can be configured to behave in a similar way to Directory Proxy Server 5.2. For information about how to configure this, see “Configuring Directory Proxy Server as a Connection Based Router” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*. For information about how to migrate to this version of Directory Proxy Server, see the *Sun Java System Directory Server Enterprise Edition 6.3 Migration Guide*.

Overview of Directory Proxy Server Features

Directory Proxy Server provides the following features:

- Manageability
  - Single point of access to directory data stored on multiple directory servers
  - Automatic referral following
  - Reactive and proactive monitoring of directory servers
  - Configuration on the command line or with a GUI
  - All connections have a normal listener port and a secure listener port
- Authentication and authorization
  - Certificate-based authentication with certificate mapping
  - Secure LDAP reverse proxy
  - LDAP control filtering
  - Proxy authorization
  - Identity mapping
  - Access control
- Distribution
  - Single point of access to a directory service spread over multiple directory servers
  - Extensible and customizable distribution algorithm
  - Server affinity to address propagation delay problem
  - Connection pooling and partial BER-decoding for performance and scalability
- Load-balancing/Fail-over
  - Routing based on the operation or the connection
  - Automatic load balancing and automatic fail over and fail back among a set of replicated LDAP directory servers
  - Three load-balancing algorithms
- Virtualization
  - Multiple virtual views for client applications
  - Aggregation of multiple heterogeneous data sources
  - Mapping of attribute names and values
  - Access to JDBC-compliant data repositories
  - Access to flat LDIF file resources

CHAPTER 16

Directory Proxy Server Load Balancing and Client Affinity

Deployments that use more than one data source to respond to client requests use load balancing to distribute work load. Client affinity can be used to reduce the risk of propagation delay in load balanced deployments.

For information about how to configure load balancing and client affinity, see Chapter 21, "Directory Proxy Server Load Balancing and Client Affinity," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

For information about how the Directory Proxy Server performs load balancing and client affinity, see the following sections:

- "LDAP Data Source Pools" on page 261
- "Load Balancing" on page 262
- "Client Affinity" on page 268

LDAP Data Source Pools

Requests from clients are distributed to an LDAP data source pool. One or more data sources are attached to the data source pool. The properties of a data source pool determine how client requests are routed to the different LDAP data sources that are attached to the pool. The following properties can be configured for an LDAP data source pool:

`client-affinity-policy`	The algorithm used to determine when client requests should exhibit affinity to a single LDAP data source
`client-affinity-timeout`	The client affinity time-out duration
`description`	A description of the LDAP data source pool
`enable-client-affinity`	A flag indicating whether or not consecutive requests from the same client should be directed to the same LDAP data source

`load-balancing-algorithm`	The algorithm used to distribute operations over load-balanced LDAP data sources

For information about how to create and configure an LDAP data source pool, see "Creating and Configuring LDAP Data Source Pools" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Load Balancing

When more than one data source is attached to a pool, load balancing determines which data source in the pool responds to the request.

For information about load balancing, see the following sections:

- "Introduction to Load Balancing" on page 262
- "Proportional Algorithm for Load Balancing" on page 263
- "Saturation Algorithm for Load Balancing" on page 264
- "Operational Affinity Algorithm for Load Balancing" on page 265
- "Failover Algorithm for Load Balancing" on page 267

Introduction to Load Balancing

Directory Proxy Server distributes requests according to a load balancing algorithm. The following load balancing algorithms can be configured:

Proportional algorithm
: Requests are distributed according to the weight of the data source and the cumulative load of the data source since the last startup of Directory Proxy Server.

Saturation algorithm
: Requests are distributed according to the weight of the data source and the number of available connections on the data source.

Operational affinity algorithm
: Requests are distributed according to the hash value. The number of hash values that are allocated to an attached data source is proportional to the weight of that data source.

Failover algorithm
: Requests are distributed exclusively to the attached data source with the highest weight for that operation.

In all load balancing algorithms, each attached data source can be configured with an independent weight for each of the following types of operation:

- Add
- Bind
- Compare
- Delete
- Modify DN
- Modify
- Search

If multiple attached data sources are configured with the same weight for a given type of operation, Directory Proxy Server distributes the requests evenly between the data sources. If a data source has a weight of `disabled` for a particular type of operation, Directory Proxy Server never distributes requests of that type to the data source. If a data source has a weight of `0` (zero) no requests are distributed to that data source.

An attached data source cannot be selected by the load balancing algorithm in the following circumstances:

- The data source is unavailable because an error occurred.
- All connections between the Directory Proxy Server and the data source are in use.

If a data source is configured as read-only, the data source cannot receive add, delete, or modify requests. The data source can receive search requests.

The load balancing algorithm works on a best-effort basis. If there are not sufficient resources for the load balancing algorithm to distribute requests by respecting weights, the weights are overruled. For example, if the number of simultaneous requests to a data source exceeds the maximum number of connections to that data source, requests are distributed to other data sources.

When the client affinity feature is active, Directory Proxy Server distributes requests by using the client affinity feature instead of using the load balancing algorithm. For information about client affinity, see "Client Affinity" on page 268.

Proportional Algorithm for Load Balancing

In the proportional algorithm, requests are distributed to attached data sources according to the following criteria:

- The type of request
- The weight of the data source as a ratio of the total weights of the other data sources in the pool
- The cumulative load since the last startup of Directory Proxy Server

After startup, the first request of a given type is distributed to the data source with the highest weight for that type of request. Directory Proxy Server continues to distribute the requests in proportion to the weight of each data source for that type of request.

If a data source becomes unavailable, Directory Proxy Server distributes the requests to remaining data sources in proportion to their weight.

The following figure illustrates how Directory Proxy Server distributes the first eight search requests to a pool of data sources with different weights. The data source with a weight of 2 processes twice as many requests as the data sources with a weight of 1.

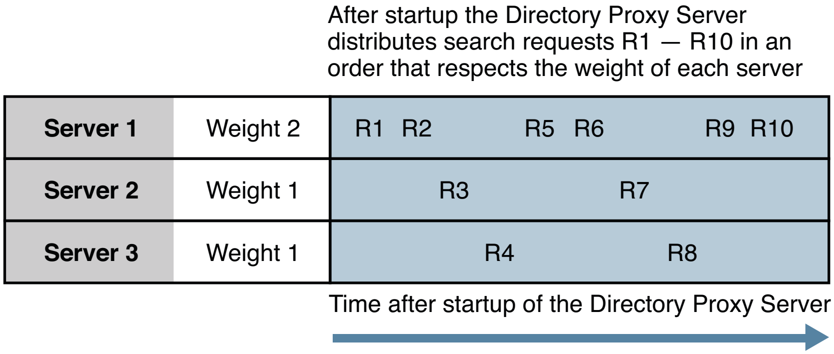

After startup the Directory Proxy Server distributes search requests R1 — R10 in an order that respects the weight of each server
Server 1
Weight 2
R1 R2
R5 R6
R9 R10
Server 2
Weight 1
R3
R7
Server 3
Weight 1
R4
R8
Time after startup of the Directory Proxy Server

FIGURE 16–1 Distribution of Requests According to the Proportional Algorithm for Load Balancing

For an example of how configure the proportional algorithm, see "To Configure the Proportional Algorithm for Load Balancing" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Saturation Algorithm for Load Balancing

In the saturation algorithm, requests are distributed to data sources according to a combination of the weight of the data source and the number of available connections.

All requests of a certain type are distributed to the data source with the highest weight, until its *saturation level* is reached. Once this level is reached, requests are distributed between this data source and the data source with the next highest weight. The saturation level is obtained by multiplying the weight of the data source by the total number of connections.

The following figure illustrates how Directory Proxy Server distributes requests to a pool of data sources with 10 connections and different weights. The number of available connections multiplied by the weight is shown in brackets.

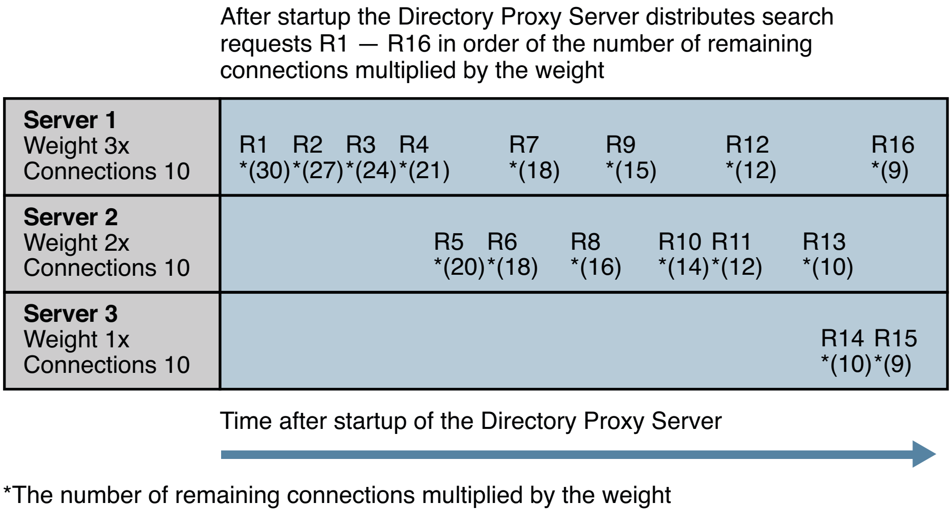

After startup the Directory Proxy Server distributes search requests R1 — R16 in order of the number of remaining connections multiplied by the weight
Server 1
Weight 3x
Connections 10
R1 R2 R3 R4 R7 R9 R12 R16
(30)(27)*(24)*(21) *(18) *(15) *(12) *(9)
Server 2
Weight 2x
Connections 10
R5 R6 R8 R10 R11 R13
(20)(18) *(16) *(14)*(12) *(10)
Server 3
Weight 1x
Connections 10
R14 R15
(10)(9)
Time after startup of the Directory Proxy Server
*The number of remaining connections multiplied by the weight

FIGURE 16–2 Distribution of Requests According to the Saturation Algorithm for Load Balancing

If your deployment includes data sources with greatly different capacity, you can use the saturation algorithm to distribute requests according to the capacity of the data source.

For an example of how configure the saturation algorithm, see "To Configure the Saturation Algorithm for Load Balancing" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Operational Affinity Algorithm for Load Balancing

In the operational affinity algorithm for load balancing, all requests are allocated a hash value according to the request type and request properties. Each hash value is allocated to an attached data source. The number of hash values that are allocated to a data source is proportional to the weight of the data source.

When a request is received, Directory Proxy Server examines the hash table to determine whether a request with that hash value has already been distributed. If the hash value already exists in the hash table, Directory Proxy Server sends the request to the data source with that hash value. If the hash value does not exist in the hash table, the request is distributed by using the proportional algorithm for load balancing.

Figure 16–3 shows an example with three attached data sources. Data source A has a weight of 3 for search operations, the other data sources have a weight of 1 for search operations. The hash table allocates $3/5_{ths}$ of the hash values to data source A, $1/5_{th}$ to data source B, and $1/5_{th}$ to data source C.

If requests have a normal range of diversity, data source A would receive three times more requests than data source B or data source C. If there is a disproportionate number of requests with identical properties, the ratio of requests between the three data sources is disturbed. For example, if a client make repeated BIND requests on the same DN, the BIND must always be serviced by the same data source.

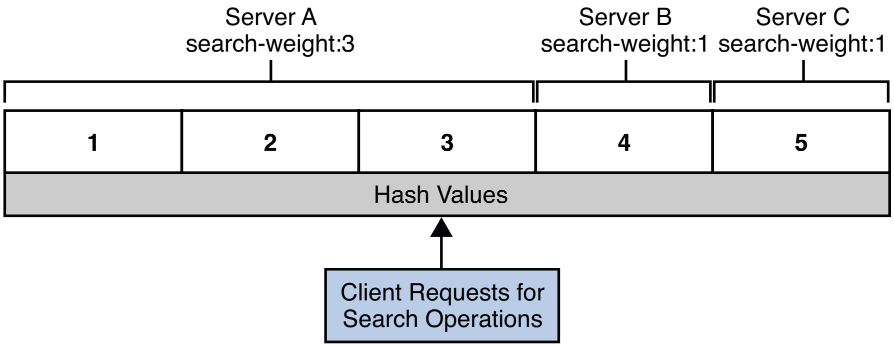


FIGURE 16–3 Distribution of Requests According to the Operational Affinity Algorithm for Load Balancing

The use of the operational affinity algorithm for load balancing is beneficial for the following features:

- Global account lockout
- Cache optimization in Directory Server

Disadvantage of Using the Operational Affinity Algorithm for Load Balancing

The operational affinity algorithm for load balancing does not ensure an evenly distributed work load across data sources.

A hash value is allocated to a request according to the type of request and the properties of the request. A range of hash values represents an arbitrary group of unrelated requests. It is possible for one range of hash values to represent many more operations than another range of hash values. A given range of hash values might represent requests that are made frequently, another range of hash values might represents requests that are almost never made.

Operational Affinity Algorithm for Global Account Lockout

By using the operational affinity algorithm for load balancing, you can ensure that the same data source always responds to bind requests from a given client. In this way, you can ensure that a client is locked out after the maximum number of failed bind attempts. If the same data source does not respond to bind requests from a given client, the client can exceed the maximum number of failed bind attempts.

When a client binds, a hash value for the request is allocated according to the bind credentials. Directory Proxy Server consults the hash table and distributes the request to the data source for that hash value. No matter how many times the client binds, the hash value is always the same. The request is always distributed to the same data source.

If a client requests a bind without the appropriate credentials, the data source rejects the bind request. If the client makes a second or third bind request, the same data source rejects the bind request. When the client exceeds the maximum number of allowed bind attempts, Directory Server locks the client out.

For an example of how configure the operational affinity algorithm for global account lockout, see “To Configure the Operational Affinity Algorithm for Global Account Lockout” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Operational Affinity Algorithm for Cache Optimization

By using the operational affinity algorithm for load balancing, searches from the same client to the same entry can always be distributed to the same data source. When a data source responds to a request, the targeted entry is stored in the cache. If the same data source responds repeatedly to the same request, the data source can benefit from using the cached data.

For an example of how configure the operational affinity algorithm for cache optimization, see “To Configure Operational Affinity Algorithm for Cache Optimization” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Failover Algorithm for Load Balancing

In the failover algorithm, requests of a given type are distributed exclusively to the attached data source with the highest weight for that operation. If that attached data source fails, requests are distributed exclusively to the attached data source with the next highest weight for that operation. If the data source with the highest weight comes back on line, requests are distributed to that data source.

For an example of how configure the failover algorithm, see “To Configure the Failover Algorithm for Load Balancing” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Client Affinity

Client affinity is defined between a client connection and a data source. When client affinity is defined, requests from a specified client connection are distributed to a specified data source in a data source pool.

The client affinity feature reduces the risk of propagation delay in deployments that use load balancing. Propagation delays can occur when a client makes consecutive requests that target the same entry if those requests are not treated by the same data source. For example, a client might make one request to change an entry and a second request to use the changed entry. If the second request is treated by a data source that has not been updated by the first request, an error occurs.

Client affinity can be configured in the following ways:

- Enabled or disabled
- Configured for all write requests after the first write request
- Configured for all requests after the first write request
- Configured for all requests after the first read request or write request
- Configured for first read request after a write request
- Configured to expire after a specified time

Client affinity takes precedence over the load balancing algorithm. Directory Proxy Server distributes a request from the specified connection to the specified data source, irrespective of the load balancing algorithm.

If client affinity is defined and enabled, the load balancing algorithm takes precedence in the following circumstances:

- The request that starts client affinity has not occurred
- The request that ends client affinity has occurred
- The client affinity time-out has expired
- The specified data source cannot be used for a request, or an error has occurred on the specified data source

A data source cannot be used for a request in the following circumstances:

- It is offline.
- It is not configured to perform the operation being requested. For example, a data source that is configured for read requests cannot respond to write requests.

For information about how to configure client affinity, see “Configuring Client Affinity” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

The client affinity feature must be used to configure Directory Proxy Server as a simple, connection based router. For information, see “Configuring Directory Proxy Server as a Connection Based Router” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

CHAPTER 17

Directory Proxy Server Distribution

Directory Proxy Server enables distribution through the definition of data views. Data views are defined with a view base, which determines the base DN of the entries in that data view. Based on the distribution algorithms provided in Directory Proxy Server, you can specify how entries are divided among the different data views.

LDAP Data Views

An LDAP data view exposes data in an LDAP server to a client request and specifies the data source pool that responds to the request. By defining LDAP data views, you can perform the following tasks:

- Expose a whole database in a single view
- Provide different views for different subtrees in a database
- Provide a unified view of different databases

There are additional types of data views but distribution can only be done with LDAP data views. For more information about other types of data views, see Chapter 18, "Directory Proxy Server Virtualization."

LDAP Data View Features

A simple LDAP data view is defined primarily by the base DN of the data view. In a simple data view all of the entries in the subtree are encompassed by the data view. Data views can exist in hierarchy, with a *superior data view* and a *subordinate data view*. A subordinate data view is a data view whose base DN is inferior to the base DN of a superior data view. The entries in a subordinate data view are excluded from the superior data view.

For information about the features of a data view, see the following sections.

- "Excluding a Subtree From a Data View" on page 272
- "Performing a Search Directed at a Superior Data View on an Excluded, Subordinate Data View" on page 272
- "Distributing Entries In a Subtree to Different Data Views" on page 274
- "Attribute Renaming and DN Renaming" on page 272

Excluding a Subtree From a Data View

When a subordinate data view is created, Directory Proxy Server automatically excludes the subordinate data view from the superior data view. When a request targets the subordinate data view, the request is sent to the subordinate data view instead of the superior data view.

By default, Directory Proxy Server automatically configures the `excluded-subtrees` parameter in the superior data view to exclude subordinate data views. For information about how to disable the automatic configuration, see "To Manually Configure the excluded-subtrees and alternate-search-base-dn Properties" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

The following subtrees are excluded by default from all data views: `cn=config`, `cn=monitor`, and `cn=proxy manager`.

Performing a Search Directed at a Superior Data View on an Excluded, Subordinate Data View

When an alternate search base is specified in a subordinate data view, search operations targeted at the superior data view are also performed in the subordinate data view.

By default, Directory Proxy Server automatically configures the `alternateSearchBase` parameter in the subordinate data view. For information about how to disable the automatic configuration, see "To Manually Configure the excluded-subtrees and alternate-search-base-dn Properties" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Attribute Renaming and DN Renaming

Each entry in a directory is identified by a DN and a set of attributes and their values. Often, the DN and the attributes defined on the client side do not map to the DN and the attributes defined on the server side.

Data views can be defined to rename DNs and attributes to values that match the server side. When a client makes a request, the DNs and attributes are renamed to match the server side. When the result is returned to a client, the DN and attributes are changed back to match the client side.

Attribute Renaming

The following figure illustrates how attribute renaming is performed by Directory Proxy Server.

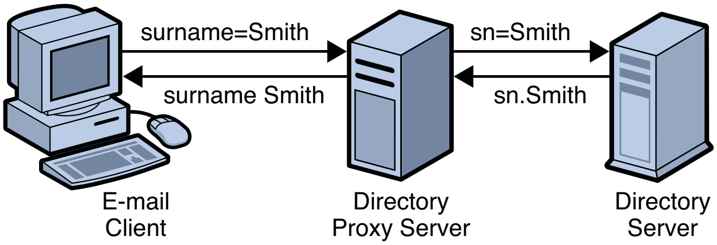


FIGURE 17–1 Attribute Renaming

In Figure 17–1, the email client expects the last names to be specified by the attribute `surname` However, in the LDAP server, last names are specified by the attribute `sn`. When attributes are renamed, only the name of the attribute is affected — the value of the attribute is not changed. However, when attributes are renamed all entries with that name are renamed.

For information about how to configure attribute renaming, see "To Configure Attribute Renaming" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

DN Renaming

The following figure illustrates how DN renaming is performed by Directory Proxy Server.

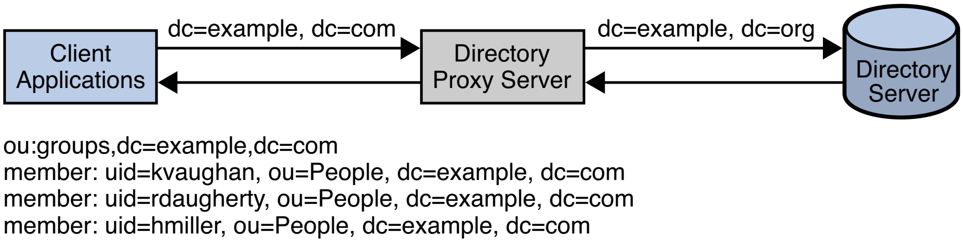

dc=example, dc=com
dc=example, dc=org
Client
Applications
Directory
Proxy Server
Directory
Server
ou:groups,dc=example,dc=com
member: uid=kvaughan, ou=People, dc=example, dc=com
member: uid=rdaugherty, ou=People, dc=example, dc=com
member: uid=hmiller, ou=People, dc=example, dc=com

FIGURE 17–2 DN Renaming

In Figure 17–2, the client contains the `dc=example, dc=com` database. The LDAP server contains the `dc=example, dc=org` database. The Directory Proxy Server renames the DNs.

Attributes that contain DNs must also be renamed if those DNs are in the portion of the DIT that is affected by the original DN renaming. In Figure 17–2, the `group` attribute contains a list of the DNs of group members. When `dc=example, dc=com` is renamed to `dc=example, dc=org`, the DNs in the group attribute must also be renamed.

For information about how to configure DN renaming, see "To Configure DN Renaming" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Distributing Entries In a Subtree to Different Data Views

A distribution algorithm distributes operations across data views that have the same base DN. The type of distribution algorithm is defined by the `distribution-algorithm` parameter.

To determine how to distribute operations, the distribution algorithm considers the value of the attribute *directly below* the base DN of the data view. For example, consider a data view with a base DN of `ou=people,dc=example,dc=com`. If a search operation contains the base DN `uid=23,ou=people,dc=example,dc=com`, the distribution algorithm considers `uid` to be the routing attribute, because `uid` is directly below the base DN of the data view. The algorithm then attempts to match the value `23` to determine how to route the operation.

However, if the search operation contains the base DN `uid=23,ou=managers,ou=people,dc=example,dc=com`, the distribution algorithm considers `ou` to be the routing attribute, because `ou` is directly below the base DN of the data view. Because `ou` does not match the `uid` specified in the search query, the distribution algorithm cannot distribute the search correctly. For distribution to work in this case, the base DN of the data view should be `ou=managers,ou=people,dc=example,dc=com`.

You must therefore ensure that the base DN of the data view is appropriate to the distribution algorithm.

The following distribution algorithms are provided with Directory Proxy Server:

Pattern matching

Requests are distributed to data views based on the match between the parameters of the requests and one or more patterns. Patterns are defined by the following parameters:

- `pattern-matching-base-object-search-filter`
- `pattern-matching-dn-regular-expression`
- `pattern-matching-one-level-search-filter`
- `pattern-matching-subtree-search-filter`

The syntax supported by the pattern matching algorithm is specified by the Java Pattern class (documented at http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html (`http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html`)). This syntax is not the same as the usual `regex` syntax.

Numeric

Requests are distributed to data views according to the numeric value of the RDN in the request. The numeric value is taken from the value of the first RDN beneath the base DN of the data view. Numeric bounds are defined by these parameters:

- `numeric-attrs`
- `numeric-default-data-view`
- `numeric-lower-bound`

- `numeric-upper-bound`

Lexicographic

Requests are distributed to data views according to the lexicographic value of the RDN in the request. Lexico bounds are taken from the value of the first RDN beneath the base DN of the data view. Lexico bounds are defined by these parameters:

- `lexicographic-attrs`
- `lexicographic-lower-bound`
- `lexicographic-upper-bound`

Replication

Requests are distributed to data views according to the role of the data view in replication. The algorithm distributes write operations to all data sources in the data source pool and read operations to a single data source. The replication role is defined by the `replication-role` parameter. A data view can have a master role or a consumer role.

You can also configure Directory Proxy Server to support your custom distribution algorithms. For more information about configuring custom distribution algorithms, see “To Configure Custom Distribution Algorithm” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

For information about how to configure a distribution algorithm, see “Data Views With Hierarchy and a Distribution Algorithm” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*. For information about the parameters used with the distribution algorithms, see `distribution-algorithm`(5dpconf).

Limitations of Distribution Algorithms

The distribution algorithms provided with Directory Proxy Server have certain limitations in specific request scenarios.

The following list outlines the situations in which requests do not respect the distribution algorithm. The examples in this list assume that the routing attribute is `uid` and the view base of the data view is `dc=example,dc=com`.

- When the search base ends with the view base and the scope is `base`, requests are always distributed to the first data view. For example:

```
$ ldapsearch -b "ou=people,dc=example,dc=com" -s base "uid=116352"
```

- When the search base ends with the view base and the scope is one level or subtree, requests are always distributed to the first data view. For example:

```
$ ldapsearch -b "ou=people,dc=example,dc=com" -s sub "uid=116352"
```

- When the search base ends with the view base and starts with the routing attribute, but the search filter does not contain the routing attribute, requests are distributed to all data views. For example:

  ```
  $ ldapsearch -b "uid=116352",ou=people,dc=example,dc=com" -s base "objectclass=*"
  ```

  In this example, requests are distributed correctly if the RDN value matches the data view criteria.
- When the search base ends with the view base and contains the routing attribute, but the search filter does not contain the routing attribute, requests are distributed to all data views. For example:

```
$ ldapsearch -b "cn=myAccount,uid=116352,ou=people,dc=example,dc=com" -s base "objectclass=*"
```

In this example, requests are distributed correctly if the RDN value matches the data view criteria.

Use Cases for Data Views

This section describes use cases for LDAP data views. All of the examples assume that the connection handler allows all client connections to be processed by Directory Proxy Server.

For examples of data views in different deployments, see the following sections:

- "Data Views to Route All Requests, Irrespective of the Target DN of the Request" on page 276
- "Data Views to Route Requests When a List of Subtrees Are Stored on Multiple, Data-Equivalent Data Sources" on page 277
- "Data Views to Provide a Single Point of Access When Different Subtrees Are Stored on Different Data Sources" on page 278
- "Data Views to Route Requests When Different Parts of a Subtree Are Stored in Different Data Sources" on page 279
- "Data Views to Route Requests When Superior and Subordinate Subtrees Are Stored in Different Data Sources" on page 281
- "Data Views With Hierarchy and a Distribution Algorithm" on page 283

Data Views to Route All Requests, Irrespective of the Target DN of the Request

This section describes a data view that routes all requests to a data source pool, irrespective of the target DN of the request. This data view is called the *root data view*. The root data view is created by default when an instance of Directory Proxy Server is created.

The example in this section has multiple data sources that contain the same set of subtrees. The data sources are data-equivalent and pooled into one data source pool for load balancing. A data view is configured with a base DN at the rootDSE, represented as “ ”. Figure 17–3 shows an example deployment.

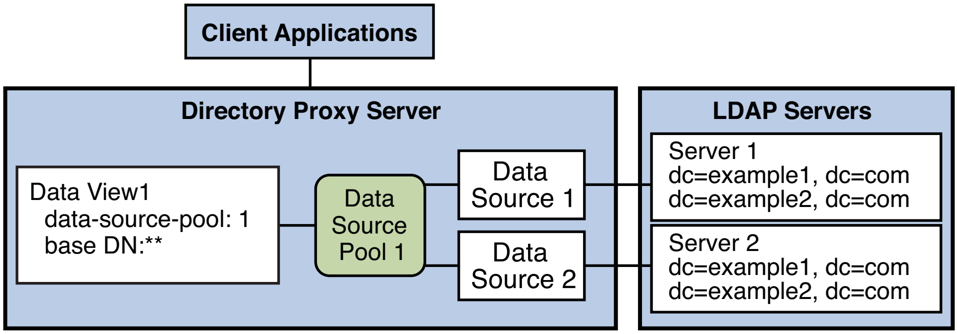


FIGURE 17–3 Example Deployment That Routes All Requests to a Data Source Pool, Irrespective of the Target DN

Because the base DN of the data view is the rootDSE, the data view encompasses the base DN of all possible requests. *All* requests are forwarded to the data source pool, irrespective of the target DN or whether the data source contains an entry for the request.

If Directory Proxy Server receives a request with a target DN that does not exist in the data source, the request is forwarded to the data source pool. The data source that responds to the request returns an error.

For information about how to configure the data view in Figure 17–3, see “Data Views That Route All Requests, Irrespective of the Target DN of the Request” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Data Views to Route Requests When a List of Subtrees Are Stored on Multiple, Data-Equivalent Data Sources

This section describes data views that route requests targeted at a list of subtrees to a set of data-equivalent data sources.

The example in this section has multiple data sources that each contain the same set of subtrees. The data sources are data-equivalent and pooled into one data source pool for load balancing. A data view is configured for each subtree to expose that subtree to client requests. Figure 17–3 shows the example deployment.

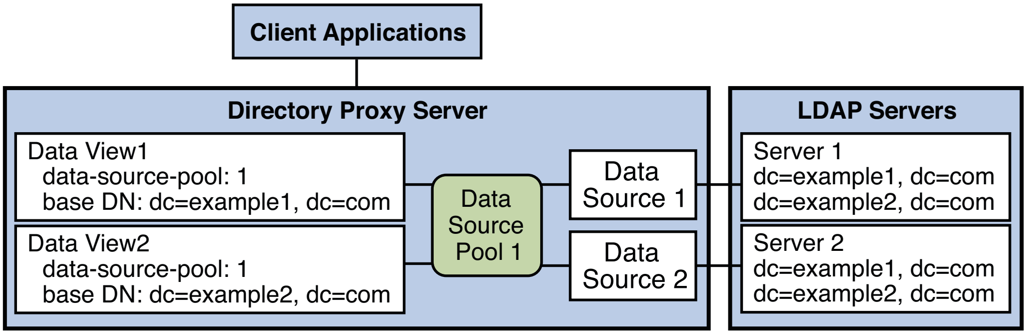


FIGURE 17–4 Example Deployment That Routes Requests When a List of Subtrees Is Stored on Multiple, Data-Equivalent Data Sources

A request is exposed to a data view only if the target DN is subordinate to the base DN of the data view. When a request is exposed to a data view, the request is forwarded to the data source pool specified by the data view.

If the target DN of a request is not subordinate to the base DN of any data view, Directory Proxy Server returns an error.

In Figure 17–4, requests that target `dc=example1,dc=com` or `dc=example2,dc=com` are forwarded to the data source pool. Directory Proxy Server returns an error for requests that target neither `dc=example1,dc=com` nor `dc=example2,dc=com`.

For information about how to configure the data views in this section, see "Data Views That Route Requests When a List of Subtrees Is Stored on Multiple, Data-Equivalent Data Sources" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Data Views to Provide a Single Point of Access When Different Subtrees Are Stored on Different Data Sources

This section describes how Directory Proxy Server provides a single point of access to different subtrees of data on multiple data sources. The example in this section contains a data view is for each subtree, to expose that subtree to client requests. A data source pool is configured for each set of data-equivalent data sources. Figure 17–5 shows the example deployment.

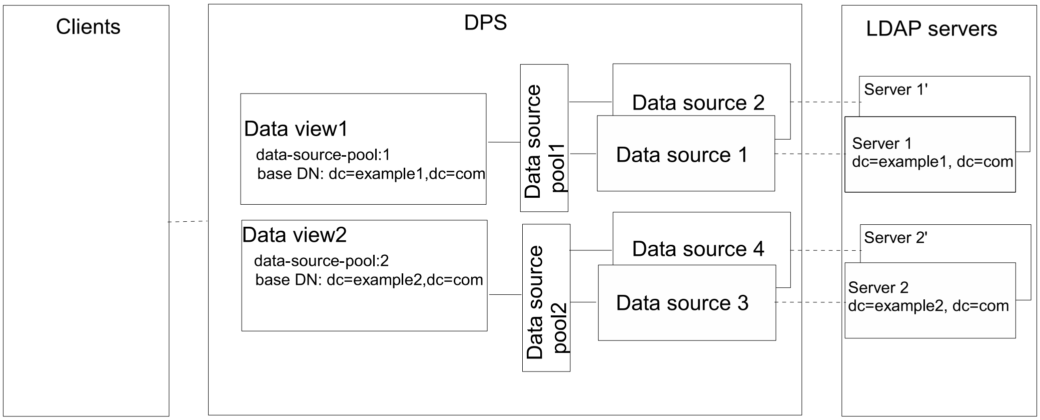


FIGURE 17–5 Example Deployment That Provides a Single Point of Access When Different Subtrees Are Stored on Different Data Sources

The Directory Proxy Server exposes a request to a data view if the DN targeted by the request is subordinate to the base DN of the data view. When a request is exposed to a data view, the request is forwarded to the data source pool specified by the data view.

If a request has a target DN that is not subordinate to the base DN of a data view, Directory Proxy Server returns an error.

In Figure 17–5, client requests that target `dc=example1,dc=com` are forwarded to the data source pool 1 and are treated by data source 1 or data source 1'. Client requests that target `dc=example2,dc=com` are forwarded to the data source pool 2 and are treated by data source 2 or data source 2'. The Directory Proxy Server returns an error for client requests that target neither `dc=example1,dc=com` nor `dc=example2,dc=com`.

For information about how to configure a data view to provide a single point of access to different subtrees stored in multiple data sources, see “Data Views That Provide a Single Point of Access When Different Subtrees Are Stored in Different Data Sources” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Data Views to Route Requests When Different Parts of a Subtree Are Stored in Different Data Sources

This section describes how Directory Proxy Server provides a single point of access to different parts of a subtree stored in multiple data sources. To route requests for different parts of a subtree, Directory Proxy Server uses a distribution algorithm. In the example in this section, Directory Proxy Server uses the numeric distribution algorithm. For more information about distribution algorithms, see "Distributing Entries In a Subtree to Different Data Views" on page 274.

The example in this section contains two data views with the same base DN. A numeric distribution algorithm is used to separate entries into different data views. A data source pool is configured for each set of data-equivalent data sources. Figure 17–6 shows the example deployment.

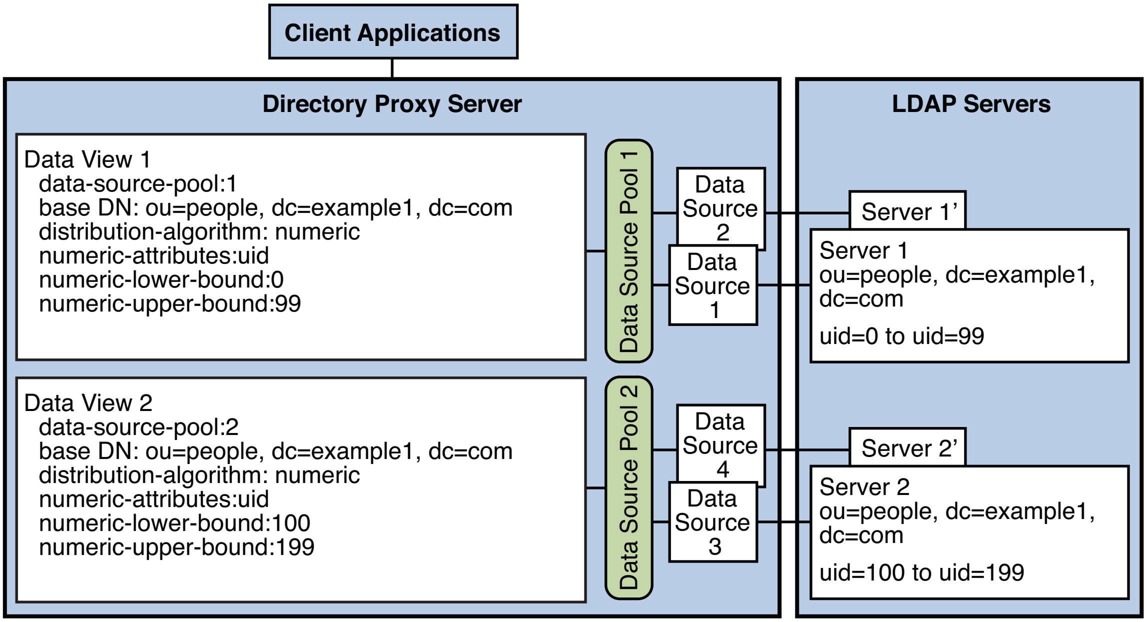


FIGURE 17–6 Example Deployment That Routes Requests When Different Parts of a Subtree Are Stored in Different Data Sources

Directory Proxy Server exposes a request to the data view which satisfies the following conditions:

- The DN targeted by the request is subordinate to the base DN of the data view
- The parameters of the requests match the pattern specified by the distribution algorithm in the data view

When a request is exposed to a data view, the request is forwarded to the data source pool specified by the data view.

If a request that does not match the conditions of any data view, Directory Proxy Server returns an error.

For information about how to configure a data view to provide a single point of access to different parts of subtree on multiple data sources, see “Data Views That Provide a Single Point of Access When Different Parts of a Subtree Are Stored in Different Data Sources” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Data Views to Route Requests When Superior and Subordinate Subtrees Are Stored in Different Data Sources

This section describes how Directory Proxy Server provides a single point of access when a superior branch of a subtree is stored in a different data source to a subordinate branch.

By default, Directory Proxy Server automatically sets the `excluded-subtrees` property and the `alternate-search-base-dn` property. However, the automatic management of the `excluded-subtrees` property and the `alternate-search-base-dn` property can be disabled. For information about how to manually configure the `excluded-subtrees` property and the `alternate-search-base-dn` property, see “To Manually Configure the excluded-subtrees and alternate-search-base-dn Properties” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

The example in Figure 17–7 contains three data views. The base DN of `dataview–1` is superior to the base DNs of `dataview-2` and `dataview-3`.

The `excluded-subtrees` property on `dataview-1` excludes `dataview-2` and `dataview-3` from `dataview-1`. The `alternate-search-base-dn` properties on `dataview-2` and `dataview-3` include `dataview-2` and `dataview-3` in search operations targeted at `dataview-1`. Figure 17–7 shows the example deployment.

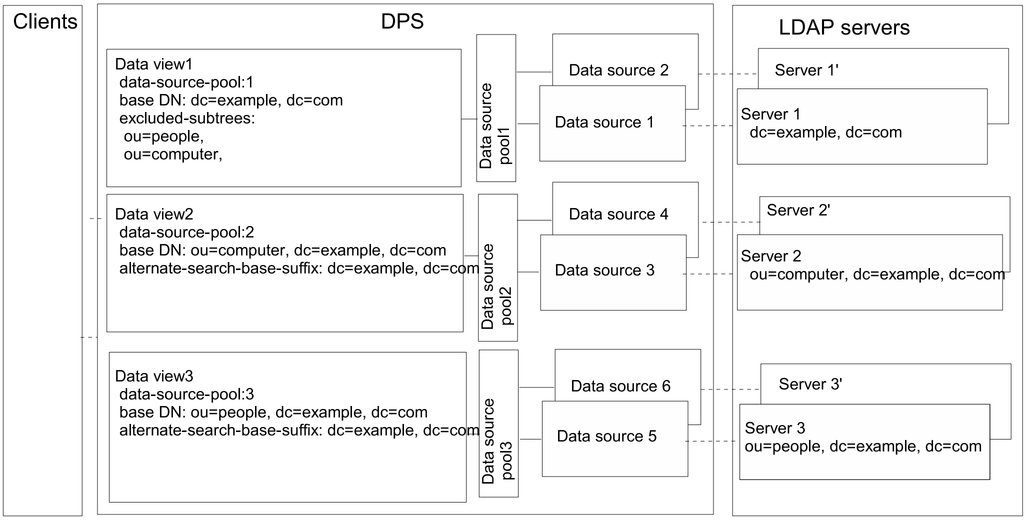


FIGURE 17–7 Example Deployment to Route Requests When Superior and Subordinate Subtrees Are Stored in Different Data Sources

Directory Proxy Server exposes a request to the data view which satisfies the following conditions:

- The DN targeted by the request is subordinate to the base DN of the data view
- The DN targeted by the request is not excluded from the data view by the `excluded-subtrees` parameter

When a request is exposed to a data view, the request is forwarded to the data source pool specified by the data view.

If a request does not match the conditions of any data view, the request cannot be exposed to a data view and Directory Proxy Server returns an error.

In Figure 17–7, client requests that target `dc=example,dc=com` but do not target `ou=computer, dc=example, dc=com` or `ou=people, dc=example, dc=com` are forwarded to the data source pool 1. Such requests are treated by data source 1 or data source 1'. Client requests that target `ou=computer, dc=example, dc=com` or `ou=people, dc=example, dc=com` are forwarded to data source pool 2 and data source 3, respectively. Directory Proxy Server returns an error for client requests that do not target `dc=example,dc=com`.

All three data views are candidates for search operations that are targeted at `dc=example,dc=com`.

For information about how to configure a data view to provide a single point of access to different parts of subtree in multiple data sources, see “Data Views That Provide a Single Point of Access When Superior and Subordinate Subtrees Are Stored in Different Data Sources” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Data Views With Hierarchy and a Distribution Algorithm

Different data views can be used in the same topology to expose or hide parts of a subtree. Figure 17–8 shows are an example with data views that combine the hierarchy shown in Figure 17–7 with the distribution algorithms shown in Figure 17–6.

The example in Figure 17–8 contains four data views. The base DN of data view 1 is superior to the base DNs of the other data views. Data view 3 and data view 4 have the same base DN, but a numeric distribution algorithm separates entries into the different data views. Figure 17–8 shows the example deployment.

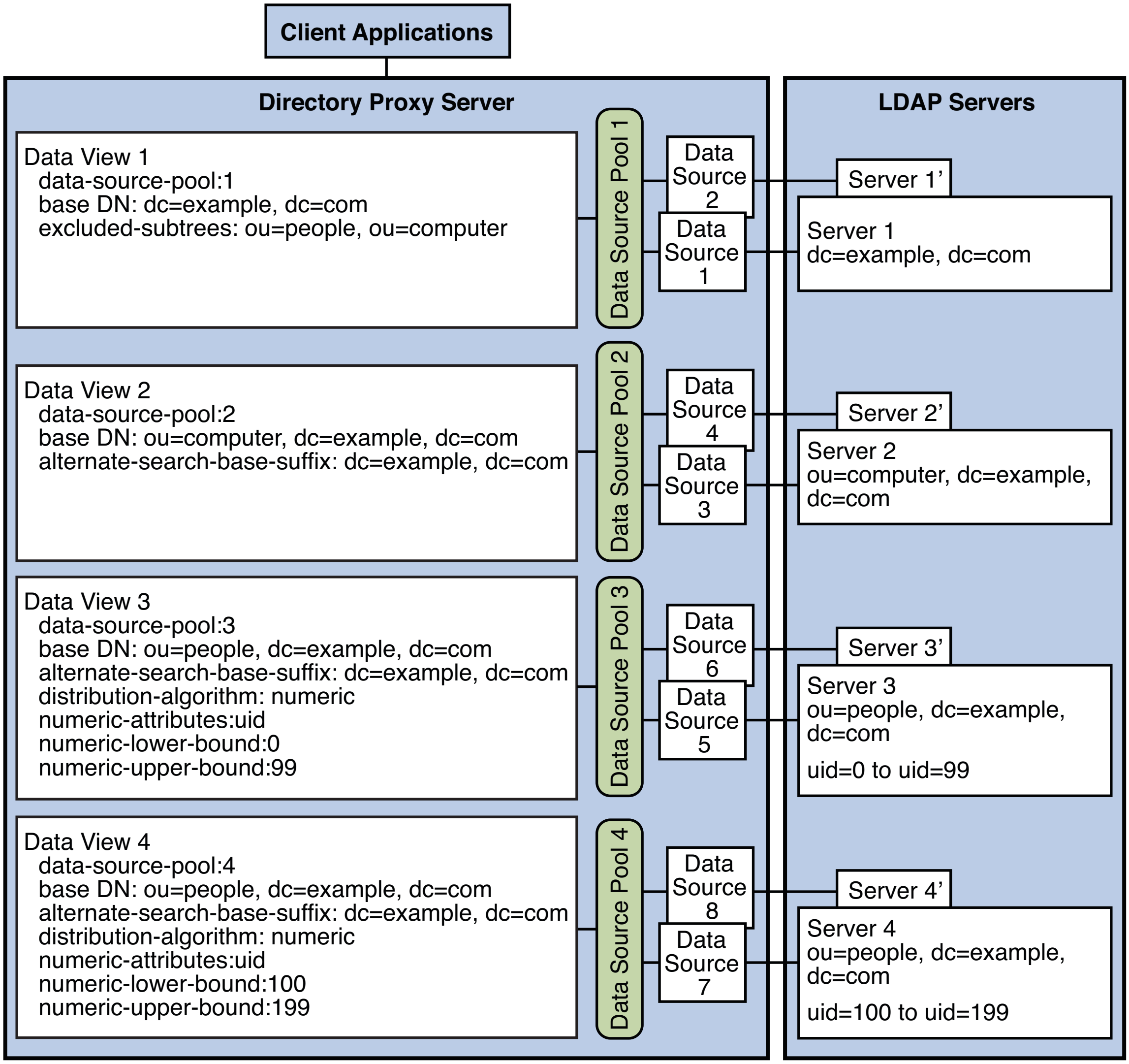


FIGURE 17–8 Data View With Hierarchy and a Distribution Algorithm

The `excluded-subtrees` property on `dataview-1` excludes the other data views from `dataview-1`. The `alternate-search-base-dn` property on `dataview-2`, `dataview-3`, and `dataview-4` includes these data views in search operations targeted at `dataview-1`.

Directory Proxy Server exposes a request to the data view which satisfies the following conditions:

- The DN targeted by the request is subordinate to the base DN of the data view

- The DN targeted by the request is not excluded from the data view by the `excluded-subtrees` parameter
- The parameters of the requests match the pattern specified by the distribution algorithm

When a request is exposed to a data view, the request is forwarded to the data source pool specified by the data view.

If a request does not match the conditions of any data view, Directory Proxy Server returns an error.

For information about how to configure a complex data view, see “Data Views With Hierarchy and a Distribution Algorithm” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

CHAPTER 18

Directory Proxy Server Virtualization

Directory Proxy Server enables virtualization through the definition of virtual data views. Virtual data views enable you to display physical data in a different way. This chapter describes how virtual data views are created, and the kinds of virtual data views that are available in Directory Proxy Server.

The chapter covers the following topics:

Construction of Virtual Data Views

A virtual data view is essentially a physical data view on which certain transformation actions have been defined. The transformation actions take place in real time, to create the virtual data view. The following figure shows how transformation actions are defined on a physical data view to create a virtual data view.

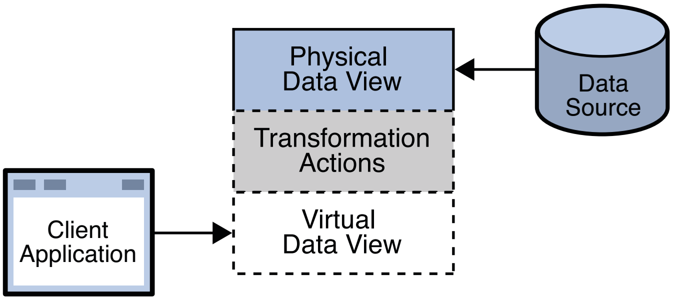


FIGURE 18–1 Virtual Data View

In addition to the transformation actions, certain properties can be defined on a data view, which restrict the way in which data can be managed through that data view. The additional virtual data view properties are described in "Additional Virtual Data View Properties" on page 298.

Caution – Virtual data views imply a performance impact. The significance of the performance impact depends on several factors including the size of the physical data source, the complexity of the transformation, and the complexity of any virtual ACIs you might use.

Virtual Data Transformations

Virtual data transformations create a virtual data view from a physical data view. Practically, you never define a virtual data view. Instead, you specify the transformations that you require and define these on an existing physical data view. A transformation performs a specific *action* in a certain *direction*. The direction of a transformation determines the transformation model. When you define a virtual data transformation, you create a virtual attribute that exists only in the context of the virtual data view.

A transformation is defined on a data view, by using the `dpconf` command as follows:

```
$ dpconf add-virtual-transformation -h host -p port -D bindDN /
 view-name model action attr-name [parameters...]
```

The *view-name* refers to the data view on which the transformation is defined. The *attr-name* refers to the virtual attribute that is created. The model, action, and additional parameters are described in the following sections.

The name of the virtual transformation can be set by using the following command:

```
$ dpconf set-virtual-transformation-prop -h host -p port -D bindDN /
 view-name transformation-name property:value [property:value]
```

Transformation Models

The transformation model is determined by the direction of a transformation, in other words, whether the transformation is applied during the request, during the response, or both.

In this sense, transformations can be categorized into the following types:

- Mapping transformations (bidirectional transformations)
- Write transformations (inbound transformations)
- Read transformations (outbound transformations)

Mapping Transformations

The most common transformation is a bidirectional (mapping) transformation. A mapping transformation is applied during the request, and its inverse is applied during the response. These transformations are called *mappings* because in effect, an attribute or entry in the physical data view maps to an attribute or entry in the virtual data view. Mapping transformations enable you to process existing values before assigning them to a DN component, an attribute type or value, or an object class.

The following diagram illustrates the principals of a mapping transformation.

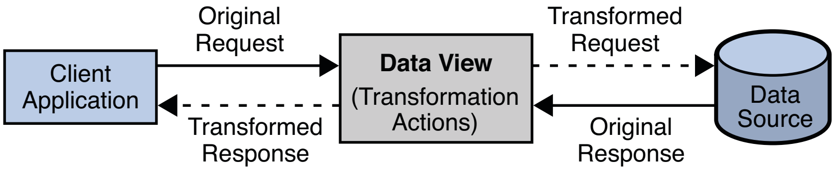


FIGURE 18–2 Mapping Transformation

A mapping transformation is defined on a data view, by running the `dpconf` command as follows:

```
$ dpconf add-virtual-transformation -h host -p port -D bindDN /
view-name mapping action attr-name [parameters]
```

EXAMPLE 18–1 When Would You Use a Mapping Transformation?

Imagine, for example, an organization has a physical data source that contains entries with the attributes `surname` and `givename`. The organization has a client application that requires entries to have a `cn` (common name) attribute of the form `givenname surname`.

The client application sends a search request for an entry of the form `cn=Carlos Fuentes`. A transformation is defined that extracts the name and surname during this request and transforms the request to one of the form `surname=Fuentes, givenname=Carlos`. The corresponding entry is located in the data source. Before returning this entry to the client application, the inverse transformation is performed. The client application receives the entry as `cn=Carlos Fuentes`, which it understands.

EXAMPLE 18–1 When Would You Use a Mapping Transformation? *(Continued)*

This request is transformed to be of the form `surname=Fuentes, givenname=Carlos`. Similarly, the client application sends a modify request to change the `cn` attribute of an entry to `Lisa Davis`. The request is transformed so that the `givenname` attribute of the physical entry is modified to `Lisa` and the `surname` attribute is modified to `Davis`.

Write Transformations

A write transformation is applied during the request, but not during the response. A write transformation *changes the physical data in storage*.

The following diagram illustrates the principals of a write transformation.

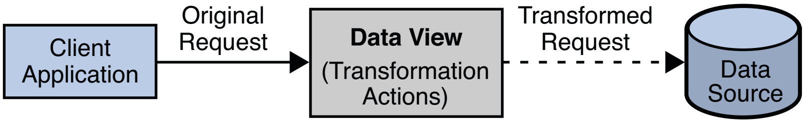


FIGURE 18–3 Write Transformation

A write transformation is defined on a data view, by using the `dpconf` command as follows:

```
$ dpconf add-virtual-transformation -h host -p port -D bindDN /
view-name write action attr-name [parameters]
```

EXAMPLE 18–2 When Would You Use a Write Transformation

Imagine an organization has a legacy application whose function is to add person entries to a data source. The application adds the entries without the `telephoneNumber` attribute. The physical data source has been upgraded and the `telephoneNumber` is now a mandatory attribute for person entries. The transformation required here is to add the `telephoneNumber` attribute during the add request. This transformation changes the entry that is written to the database. No reverse transformation is required.

Read Transformations

A read transformation is applied only during the response to a request. No transformation is applied during the request and the physical data is not changed.

The following diagram illustrates the principals of a read transformation.

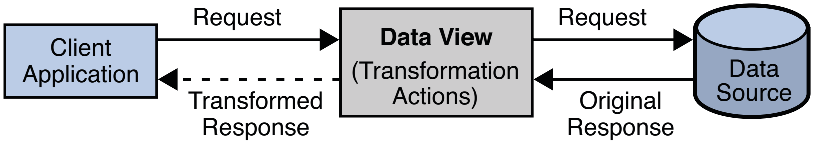


FIGURE 18–4 Read Transformation

A read transformation is defined on a data view, by using the `dpconf` command as follows:

```
$ dpconf add-virtual-transformation -h host -p port -D bindDN /
view-name read action attr-name parameters
```

EXAMPLE 18–3 When Would You Use a Read Transformation

Imagine an organization has a legacy application whose function is to display person entries. The application does not support entries that do not contain a `mail` attribute. The physical data source has been upgraded and the `email` attribute no longer exists for person entries (e-mail addresses are constructed using other attributes).

The transformation required here is to add the `mail` attribute during the search response. This transformation changes the entry that is read from the database and adds a `mail` attribute whose value is *givenname*`.`*surname*`@example.com`. No reverse transformation is required and the physical data is not changed.

Note that with the above transformation, the `mail` attribute makes no sense in a search request filter. Search request filters must contain physical attributes.

Transformation Actions

Transformation actions describe what a transformation does to its target entry or entries. The following transformation actions are possible:

- **Construct an attribute.** This action enables you to construct a virtual attribute that does not actually exist in the physical data source but is required by a client application. The action can also be used to alter an add or modify request to construct an attribute that is required by the physical data source.

  To construct the attribute, use the `add-attr` transformation action.
- **Remove an attribute.** This action enables you to delete an attribute from a client request if that attribute is not permitted by the schema on the physical data source. The action can also be used to remove an attribute from the response sent to a client application if the client application does not require that attribute.

  To remove an attribute, use the `remove-attr` transformation action.
- **Construct an attribute value.** This action enables you to create an attribute value from other attribute values.

To create an attribute value, use the `add-attr-value` transformation action.

- **Delete an attribute value.** This action enables you to remove the value from an attribute. It is usually used to remove one or more values from a multi-valued attribute if either the client application or the data source schema does not permit multi-valued attributes.

  To remove an attribute value, use the `remove-attr-value` transformation action.
- **Add a default value to an attribute.** This action enables you to add a default value to an attribute, if no value exists.

  To add a default value to an attribute, use the `def-value` transformation action.
- **Map one attribute value to another.** This action enables you to have two different values for an attribute, depending on whether the attribute is being written to a data source or returned to a client application.

  To map attribute values, use the `attr-value-mapping` transformation action, with the `internal-value` and `view-value` parameters.

Note – Directory Proxy Server supports two ways of mapping attribute values — simple attribute mapping and mapping through a virtual transformation. In general, attribute mapping is simpler to configure and slightly better in terms of performance. For more information, see "Renaming Attributes and DNs" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

The results of a transformation action depend on the transformation model.

Transformation Parameters

Transformation parameters provide the value of a virtual attribute. This value can either be a default value, or rule that creates the value from other attribute values.

The following transformation parameters are accepted:

- *value*. This parameter is applied to all transformation actions that add an attribute value, other than the `attr-value-mapping` action.
- internal-value:*value*. This parameter applies only to the `attr-value-mapping` action, and to the `remove-attr-value` action when used with the `mapping` model. It describes the value of the attribute that is written to or read from the physical data source.
- view-value:*value*. This parameter applies only to the `attr-value-mapping` action, and to the `remove-attr-value` action when used with the `mapping` model. It describes the value of the attribute that is returned to or sent by the client application.

Transformation parameters take the following syntaxes:

- **Constant.** Used to generate an attribute with a static default value.

  For example, the parameter `0800–5994654` might be used to provide a default telephone number.
- **Attribute value.** Used to create a new attribute from an existing attribute in the entry that is being processed.

  For example, the parameter `\${cn}` specifies that the value of the new attribute must be taken from the value of the `cn` attribute The escape character is required before the `$`.
- **Constant and attribute value.** Used to create a new attribute by combining an existing attribute and a static value.

  For example, the parameter `\${cn}@example.com` specifies that the value of the new attribute must be taken from the value of the `cn` attribute and a static domain name.
- **Macro.** Used to create an attribute by manipulating the value of an existing attribute.

  The macro is a Java regular expression. For more information about Java regular expressions, see `http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html`.

  The following macros are supported:

  - Increase the value of an attribute by a consistent amount:

    `increment(`*source-attribute-value*`,`*increment*`)`

    For example, the macro `increment(\$(uid),10)` specifies that the value of the new attribute is obtained by adding 10 to the value of the `uid` attribute present in the entry.
  - Decrease the value of an attribute by a consistent amount:

    `decrement(`*source-attribute-value*`,`*decrement*`)`

    For example, the macro `decrement($(uid),10)` specifies that the value of the new attribute is obtained by subtracting 10 from the value of the `uid` attribute present in the entry.
  - Use part of an existing attribute value.

    `substring(`*source-attribute-value*`,`*begin-index*`[,`*end-index*`])`

    The `begin-index` is inclusive and the `end-index` is exclusive. That is, the substring begins on the character specified by the `begin-index` and ends on the character just before the `end-index`.

    For example, to create a new attribute whose value is the value of the `cn` attribute minus the first two characters, you would define the following macro:

    ```
    substring(\${cn},2)
    ```

To create a new attribute whose value contains only the first two characters of the value of the `cn` attribute, you would define the following macro:

```
substring(\${cn},0,2)
```

- Use part of an existing attribute value by splitting that value at a certain point.

  `split(`*source-attribute-value*`,`*token-index*`,`*regular-expression*`)`

  For example, the macro `split\(\${mail},1,"@"\)` returns the domain.

Note – The transformation parameter syntax is slightly different when used in the context of a join data view. For more information, see "Virtual Data Transformations on Join Data Views" on page 302.

Transformation Examples

The following sections provide use cases in which virtual data views are required, and the combination of transformation models and actions required to implement the use cases.

EXAMPLE 18–4 Adapting an ADAM Object Class For LDAP Compliance

An organization, Example A, stores its users in an LDAP directory. Example A acquires another company, Example B, which stores its users in an ADAM directory.

In Example A's LDAP directory, a user is stored as an `inetOrgPerson`. In Example B's directory, a user is stored as a `user`. A transformation is required that maps the ADAM `user` object class to the LDAP `inetOrgPerson` object class.

The following transformation is defined on the physical data view of Example A's directory:

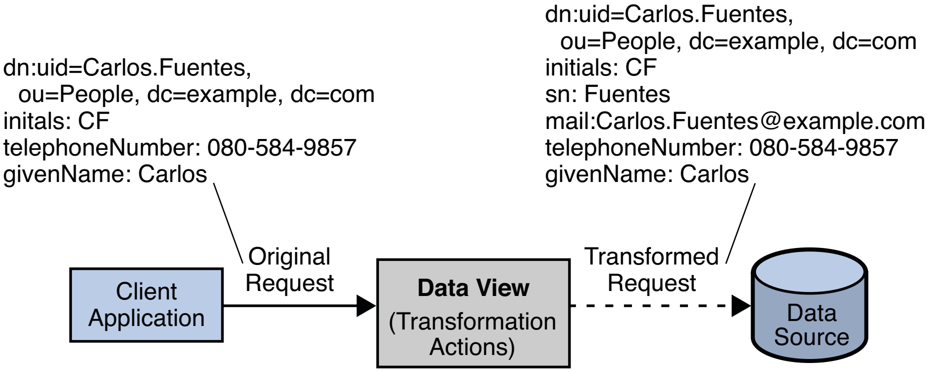
```
$ dpconf add-virtual-transformation -h myHost -p 2389 -D "cn=Proxy Manager" \
 exampleB-view-name mapping attr-value-mapping objectclass internal-value:user \
 view-value:inetOrgPerson
```

EXAMPLE 18–5 Constructing an Attribute With a Write Transformation

Example A stores user entries in its directory. All user entries require a `mail` attribute. If user entries without a mail attribute are added, a schema violation error is returned. Example A has a client application that adds user entries to the directory. Some user entries do not contain a mail attribute and the client application is incapable of generating one. To avoid schema violations when a user entry is added, a transformation is defined that adds the mail attribute to an add request. The value of the `mail` attribute is taken from the `uid` provided in the client add request, with the addition of `@example.com`.

EXAMPLE 18–5 Constructing an Attribute With a Write Transformation *(Continued)*

The following diagram indicates the transformation that occurs on an add request.



This transformation is defined on the physical data view by using the following `dpconf` command.

```
$ dpconf add-virtual-transformation -h myHost -p 2389 -d "cn=Proxy Manager" \
 exampleA-view-name write add-attr mail \${uid}@example.com
```

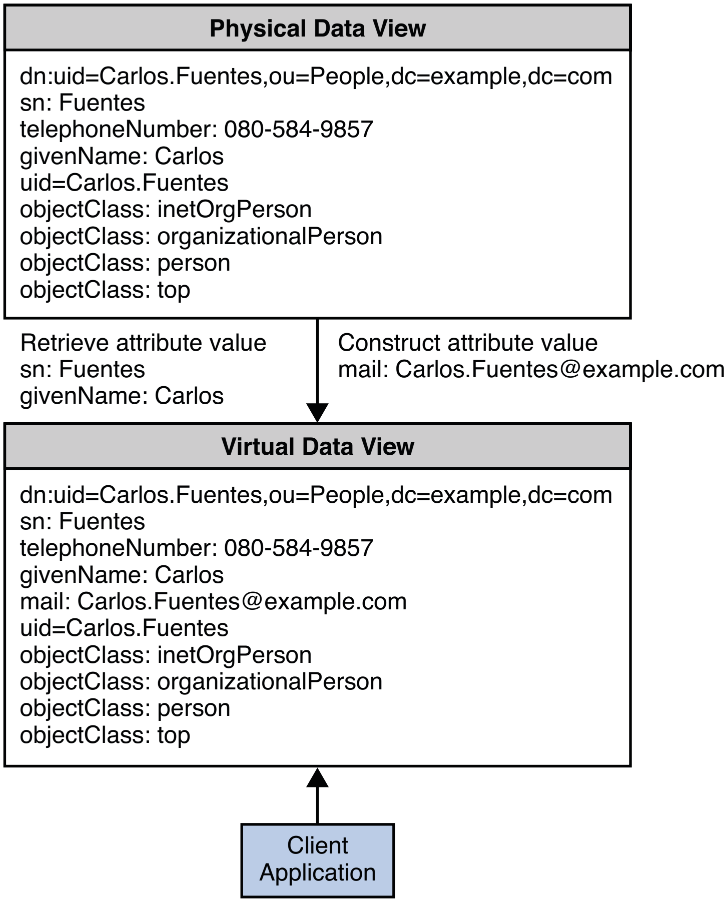
In this command, `\${uid}` means the value of the `uid` attribute for that entry.

EXAMPLE 18–6 Constructing an Attribute With a Read Transformation

Example A does not store the mail addresses of its users in its directory. However, a new client application requires that a user's mail address be returned with the user entry.

All mail addresses in the organization take the form *firstname*`.`*lastname*`@example.com`. The organization defines a virtual view in which the mail attribute is added to each user entry for reads only. The value of the mail attribute is generated by taking the value of the `givenName` and `sn` attributes that already exist in the user entry.

The following diagram indicates the transformation that occurs on user entries when they are returned in a search.


Physical Data View
dn:uid=Carlos.Fuentes,ou=People,dc=example,dc=com
sn: Fuentes
telephoneNumber: 080-584-9857
givenName: Carlos
uid=Carlos.Fuentes
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
Retrieve attribute value
sn: Fuentes
givenName: Carlos
Construct attribute value
mail: Carlos.Fuentes@example.com
Virtual Data View
dn:uid=Carlos.Fuentes,ou=People,dc=example,dc=com
sn: Fuentes
telephoneNumber: 080-584-9857
givenName: Carlos
mail: Carlos.Fuentes@example.com
uid=Carlos.Fuentes
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
Client Application

EXAMPLE 18–6 Constructing an Attribute With a Read Transformation *(Continued)*

This transformation is defined on the physical data view by using the following `dpconf` command.

```
$ dpconf add-virtual-transformation -h myHost -p 2389 -d "cn=Proxy Manager" \
 exampleA-view-name read add-attr mail \${givenname}.\${sn}@example.com
```

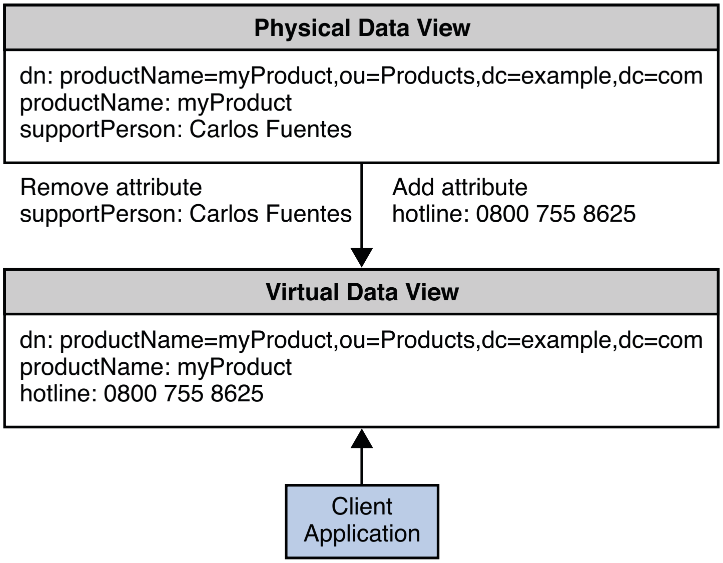
EXAMPLE 18–7 Adding a Default Attribute Value

Example A stores a number of products in its directory. In the past, each product was associated with a support person, an employee responsible for handling all support calls for that product. In the physical data store, each product is therefore associated with a `supportPerson` attribute, whose value is the DN of an employee in the organization.

EXAMPLE 18–7 Adding a Default Attribute Value *(Continued)*

The organization has changed its business process for support queries and now sends all product queries to a central hotline. To handle this change without changing the physical data, the organization defines a virtual data view where all product entries do not have a `supportPerson` attribute, but have a `hotline` attribute instead. The value of the `hotline` attribute is an 0800 number that is the same for all products.

The following diagram indicates the transformation that occurs on product entries when they are returned in a search.



This transformation is defined on the physical data view by using the following `dpconf` commands:

```
$ dpconf add-virtual-transformation -h myHost -p 2389 -d "cn=Proxy Manager" \
 exampleA-view-name read remove-attr supportPerson
$ dpconf add-virtual-transformation -h myHost -p 2389 -d "cn=Proxy Manager" \
 exampleA-view-name read add-attr hotline "0800755 8625"
```

EXAMPLE 18–8 Using a Virtual Transformation to Rename a DN

Example A has a client application that needs to sort entries according to their object class.

To do this, Example A defines a virtual transformation that rewrites the RDN of entries to include the object class of the entry along with its `cn`, whenever an entry is returned to that specific client application.

EXAMPLE 18–8 Using a Virtual Transformation to Rename a DN *(Continued)*

The following transformation is defined on the physical data view of Example A's directory:

```
$ dpconf add-virtual-transformation -h myHost -p 2389 -d "cn=Proxy Manager" \
 exampleB-view-name mapping attr-value-mapping dn internal-value:cn=\${cn} \
 view-value:cn=\${cn},objectclass=\${objectclass}
```

Additional Virtual Data View Properties

In addition to the transformation actions described previously, certain properties can be defined on a data view, which restrict the way in which data can be managed through that data view. These properties essentially provide a list of the attributes that can be read or modified through the virtual data view.

The following additional properties can be defined on a data view to present a restricted virtual data view:

- **Non-viewable attributes.** A list of the attributes that cannot be read through this data view. This list is specified by adding the multi-valued property `non-viewable-attr` to the data view. This property should be used if the number of attributes that cannot be read is small.
- **Non-writable attributes.** A list of the attributes that cannot be added or modified through this data view. This list is specified by adding the multi-valued property `non-writable-attr` to the data view. This property should used if the number of attributes that cannot be added or modified is small.
- **Viewable attributes.** A list of the attributes that can be read through this data view. This list is specified by adding the multi-valued property `viewable-attr` to the data view. This property should used if the number of attributes that can be read is small.
- **Writable attributes.** A list of the attributes that can be added or modified through this data view. This list is specified by adding the multi-valued property `writable-attr` to the data view. This property should used if the number of attributes that can be added or modified is small.

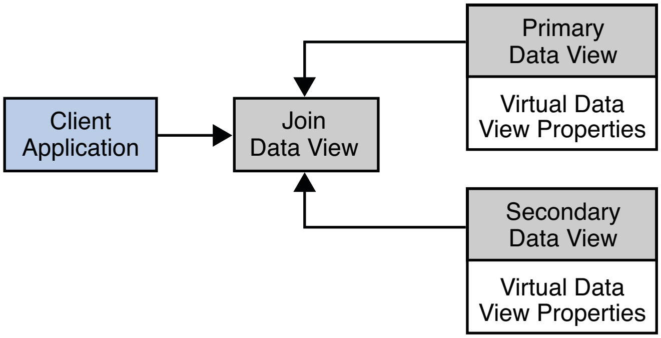
Non-viewable attributes and viewable attributes are mutually exclusive. Similarly, non-writable attributes and writable attributes are mutually exclusive.

Join Data Views

A join data view is an *aggregation* of multiple data views. The current release of Directory Proxy Server supports the aggregation of *two* data views into one join data view.

A join data view is created by specifying its name and the two existing data views that will be aggregated. One of these existing data views is considered the *primary* data view, and the other the *secondary* data view. Before you create the join data view, you need to configure the rules on the secondary data view that determine how the data is aggregated.

The following figure shows the aggregation of a primary and secondary data view to form one join data view.



Primary and Secondary Data Views

The hierarchical organization of the sources for a join data view enables Directory Proxy Server to make default decisions where the data from the primary and secondary data views do not match.

The primary data view controls the existence of entries in the join data view. The secondary data view provides supplementary data for this list of entries. In other words, if an entry exists in the secondary data view but not in the primary data view, it does not appear in the join data view.

The primary data view is the authoritative source by default. When an attribute is present on both source data views but has a different value on each, a multi-valued attribute is returned. This behavior is configurable, however. For example, you can choose to accept only the value in the primary data view, or only the value in the secondary data view.

Additional Secondary Data View Properties

In addition to the virtual data view properties described in "Additional Virtual Data View Properties" on page 298, certain properties can be defined only on a secondary data view. These properties determine how data from the two views is aggregated and how requests to the data views are handled. The following sections describe these additional properties.

Join Rules

Join rules determine how an entry from a secondary data view relates to an entry from a primary data view. Join rules are not mandatory on a secondary data view. However, if no join rule is defined, the secondary data view is not queried during LDAP operations. Directory Proxy Server provides two types of join rules, DN join rules and filter join rules.

DN Join Rules

A DN join rule determines the DN of entries in the secondary data view. A DN join rule is configured on the secondary data view by using the `dn-join-rule` property. Only one DN join rule can be configured on a secondary data view. If a DN join rule is configured on a data view, a filter join rule cannot be configured on that data view.

A DN join rule has DN syntax and can take one of the following forms:

- The DN of the secondary entry is constructed from an attribute in the primary entry.

  For example, the following DN join rule stipulates that the DNs of entries in the secondary data view should include the `cn` from the primary data view, plus the `ou=people` suffix.

  ```
  cn=\${primary-data-view.cn},ou=people
  ```

  The DN must *not* contain the base DN of the secondary data view. In this sense, it is a *relative DN*.
- The DN of the secondary entry is the same as the DN of the primary entry.

  The syntax of such a join rule is as follows:

  ```
  \${primary-data-view.dn}
  ```

  In this case, the portion of the primary and the secondary DNs below the base DN are identical, although the full DNs may differ. Imagine, for example, that the primary data view has a base DN of `o=primary` and the secondary data views has a base DN of `o=secondary`. A join rule of `\${`*primary-data-view*`.dn}` implies that the DITs below the base DN are identical. So, the entry `uid=1,o=secondary` would be associated with `uid=1,o=primary`.

Filter Join Rules

A filter join rule defines the relationship between the primary and secondary data views. A filter join rule is configured on the secondary data view by using the `filter-join-rule` property. This rule indicates how an entry should be retrieved from the secondary data view based on something in the primary data view.

Only one filter join rule can be configured on a secondary data view. If a filter join rule is configured on a data view, a DN join rule cannot be configured on that data view. A filter join rule takes the form of a filter that is used to construct an attribute from one or more attributes from the primary data view.

For example, the following filter join rule stipulates that an entry be retrieved if the entry `uid` in the primary data view matches the entry `uid` in the secondary data view.

```
uid=\${primary.uid}
```

Handling of Shared Entries

The `contains-shared-entries` property determines what should be done if an entry in the secondary data view is used by more than one entry in the primary data view.

Imagine for example, that the primary data view contains a list of user entries and the secondary data view contains a list of department numbers. A single department number in the secondary data view might apply to more than one user in the primary data view. If a user is deleted from the primary data view, you do not necessarily want that user's department number to be deleted from the secondary data view.

The `contains-shared-entries` property is set on the secondary data view only. This property is set to `TRUE` by default. This means that deleting an entry in the primary data view will not result in the deletion of the shared entry in the secondary data view. Adding an entry to the primary data view will only add the entry to the secondary data view if it does not already exist.

Handling of Binds

The `process-bind` property specifies whether a bind can be performed on the secondary data view.

By default, primary data views allow binds and secondary data views do not. The `process-bind` property is not set by default. If this property is set to `true` on a secondary data view, binds are permitted on that data view.

How Directory Proxy Server Handles Read and Write Operations to Join Data Views

If an attribute exists on both the primary and secondary data view, the attribute values are merged by the join data view. For read operations, this implies that a multi-valued attribute is returned, with the values from both data views. For write operations, the proxy queries both data views and determines where to write the value based on the content of the write operation.

If one backend data source fails during an add operation Directory Proxy Server performs an automatic rollback. The roll back takes the form of a delete operation on the data source that did not fail. This ensures the consistency of the data between the two data sources. If a roll back cannot be performed, an error is logged and an optional administrative alert is raised. Automatic roll back is on by default. You can configure automatic roll back by setting the `revertAddOnFailure` attribute to `off` (directly in `cn=config`).

If one backend data source fails during a delete operation, no roll back is performed. An error is logged and an optional administrative alert is raised.

Virtual Data Transformations on Join Data Views

Virtual data transformations are described in "Virtual Data Transformations" on page 288. The syntax of a transformation parameter differs slightly if the data transformation is defined on a join data view. Because an attribute can be obtained from more than one data view, variables that define the attribute content must be *fully qualified*. That is, the source attribute value must include the name of the data view from which the attribute is taken.

For example, the following parameter creates an attribute from existing attributes in both the primary and secondary data views:

```
\${primaryDataView.firstName}.\${secondaryDataView.lastName}@\${primaryDataView.domainName}
```

The `firstName` and `domainName` attributes are taken from the primary data view, and the `lastName` attribute is taken from the secondary data view.

LDIF Data Views

An LDIF data view is a simple virtual data view in which an LDIF file is made to look like an LDAP data source. An LDIF data view is defined by using the `dpconf` command as follows:

```
dpconf create-ldif-data-view VIEW_NAME LDIF_FILE_NAME SUFFIX_DN
```

No additional transformations are required. Directory Proxy Server automatically performs the transformations required to make the LDIF data look like LDAP data to client applications.

For information about creating and configuring LDIF data views, see "Creating and Configuring LDIF Data Views" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

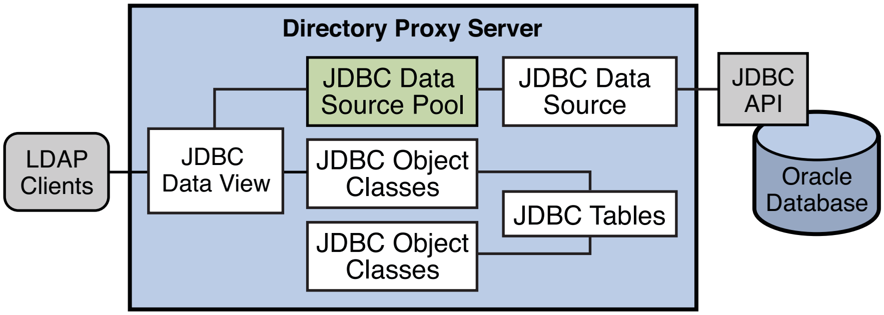
JDBC Data Views

A JDBC data view enables you to make a relational database accessible to LDAP client applications. The following configuration objects are required to set up a JDBC data view:

- **JDBC data source.** Defined for each relational database. Currently, only one JDBC data source is supported per JDBC data view.
- **JDBC data source pool.** Defined for each JDBC data source.
- **JDBC data view.** Aggregates JDBC object classes into a single data view accessible by LDAP client applications.

- **JDBC object class.** Maps one or more JDBC tables to an LDAP object class.
- **JDBC table.** Defined for each relational database table.
- **JDBC attribute.** Defines an LDAP attribute from a specified column in a JDBC table.

The following diagram shows how an LDAP client application is able to view an Oracle database in the format of an LDAP DIT, through the configuration of the JDBC objects described previously. These objects are discussed in more detail in the following sections.



An LDAP client application can also *bind* to a JDBC data view, or to a join data view that includes a JDBC data view. In this case Directory Proxy Server obtains the password from the JDBC database to do the password check. The password can be obtained in clear, SHA, or SSHA.

JDBC Data Sources and Data Source Pools

A JDBC data source is defined for each relational database. The properties of a JDBC data source include the name and location of the relational database, and the user name and password required to access the database. For a complete list of the properties that can be set for a JDBC data source, run the following command:

```
$ dpconf get-jdbc-data-source-prop -h myHost -p 2389 -d "cn=Proxy Manager"\
 jdbc-data-source-name
```

Currently, only one JDBC data source is supported for each JDBC data view. In other words, you cannot load balance across JDBC data sources.

Like LDAP data sources, JDBC data sources are organized into data source pools. The properties of a JDBC data source pool are similar to those of an LDAP data source pool. For more information about LDAP data source pools see “LDAP Data Source Pools” on page 261.

Note – Directory Proxy Server relies on metadata retrieved from the relational database. This metadata is read when Directory Proxy Server starts, or when a new JDBC data view is added. The metadata is not reread each time Directory Proxy Server processes a request. If you change the metadata in the relational database, you must restart Directory Proxy Server to take the changes into account.

The metadata is changed when any of the following changes are made:

- Changes to the structure of the database (adding or removing tables, rows, or columns)
- Changes to the case sensitivity of any column in a table

JDBC Object Classes

A JDBC object class maps an LDAP object class to one or more relational database tables. A JDBC object class works in a similar way to a join data view (see "Join Data Views" on page 299). Just as a join data view has primary and secondary source data views, a JDBC object class can obtain its information from more than one table. One table must be defined as the primary table, and additional tables, if they exist, are defined as secondary tables. The primary table controls the list of entries and additional information on these entries is extracted from the secondary tables.

When you define a JDBC object class, you must specify the following operands:

- The name of the JDBC data view to which this object class is attached.
- The name of the JDBC object class.
- The primary JDBC table from which the object class will obtain its list of entries.
- A DN pattern that controls how DNs are constructed in the data view.
- Optionally, one or more secondary JDBC tables.

JDBC Tables

A JDBC table must be created for each relational database table that will be used in the JDBC data view. When you create a JDBC table you specify the name of the table in the relational database, and the name you want to assign to this table in the JDBC data view.

The following properties apply to JDBC tables:

- **SQL table.** (`sql-table`) Specifies the name of the relational database table.

  This value must be specified when you create the JDBC table but can be changed if the SQL table name changes.
- **Single row table.** (`is-single-row-table`) Specifies that an LDAP entry has only one matching row in the relational database table.

  Generally, performance is improved if this property is set to `true` because there is no ordering in the SQL request.
- **Shared entries.** (`contains-shared-entries`) This property determines what should be done if a row in a secondary table is used by more than one entry in the primary table.

  Imagine for example, that the primary table contains a list of user details and the secondary table contains department numbers. A single department number in the secondary table might apply to more than one user in the primary table. If a user is deleted, you do not necessarily want that user's department number to be deleted from the secondary table.

  The `contains-shared-entries` property is set on secondary tables only. If this property is set to `TRUE`, deleting an LDAP entry will result in deletion of the user in the primary table but not in the deletion of the corresponding row in the secondary table.
- **Filter join rule.** (`filter-join-rule`) A filter join rule defines the relationship between primary and secondary tables.

  A filter join rule is mandatory on secondary tables, and indicates how an entry should be retrieved from the secondary table based on something in the primary table.

  Only one filter join rule can be configured on each secondary table. A filter join rule takes the form of a filter that is used to construct an LDAP attribute.

  For example, the following command creates a filter join on the secondary `phone` table. This rule stipulates that an entry be retrieved from the `phone` table if the `user_id` field in that table matches the `id` field in the `employee` table.

  ```
  $ dpconf set-jdbc-table-prop -h myHost -p 2389 -d "cn=Proxy Manager" \
   phone filter-join-rule:user_id=\${employee.id}
  ```

JDBC Attributes

JDBC attributes map LDAP attributes to entries in relational database tables. The definition of a JDBC attribute includes the name of the LDAP attribute, and the table and column in which the corresponding information is located.

For example, the following command maps the `employeeNumber` attribute to the `ID` field of the `EMPLOYEE` table.

```
$ dpconf add-jdbc-attr -h myHost -p 2389 -d "cn=Proxy Manager" \
 EMPLOYEE employeeNumber ID
```

The following properties apply to JDBC attributes:

- **LDAP syntax.** (`ldap-syntax`) This property defines the syntax used to construct the LDAP attribute from an entry in the relational database table.

  Changes to JDBC attribute syntax require a server restart before they are taken into account.
- **SQL column.** (`sql-column`) The column in the relational database table from which the LDAP attribute is obtained.
- **SQL syntax.** (`sql-syntax`) This property defines the syntax used to construct an entry in the relational database table from an LDAP entry.

Case Sensitivity in JDBC Data Views

In some cases, the LDAP attribute might be *case insensitive*, while the corresponding column in the relational database is *case sensitive*. Directory Proxy Server handles this by adding an `UPPER` keyword to equality and substring indexes. This can have serious performance implications. If the relational database requires case-sensitivity, you should therefore create specific indexes on the upper case values.

Access Control On Virtual Data Views

In a virtual data view, Directory Proxy Server exposes virtual data. Directory Proxy Server is therefore responsible for controlling who can access that data, and what parts of the data can be accessed. To control access to virtual data, you can define virtual ACIs. When Directory Proxy Server receives a request on a virtual data view, it uses the virtual ACIs, and any authentication information provided by the user, to allow or deny access to the information that is requested.

This section describes the syntax and architecture of virtual ACIs. For information about configuring virtual ACIs, see “Defining Access Control on Virtual Data Views” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Virtual ACI Definition

Virtual ACIs are defined by using the `dpsaci` operational attribute. The `dpsaci` attribute is multi-valued. This means that several ACIs can be defined for the same portion of a directory.

Directory Proxy Server is responsible for the management of the `dpsaci` attribute. This attribute can be configured along with the physical data but it is not stored with the data. When the `dpsaci` attribute is included in a request, Directory Proxy Server extracts it from the request and manages it in a dedicated ACI repository, through its own ACI data view.

A modify request that targets a virtual data view and contains the `dpsaci` attribute is effectively split into two requests by Directory Proxy Server. The first request handles only the virtual data, and the second request handles the virtual ACI.

Note – By default, write operations are forbidden on non-LDAP data views.

Global ACIs

Global ACIs are defined in the entry `cn=`*data-source-name*`,cn=virtual access controls`. These ACIs are evaluated by an ACI engine to deny or allow requests from a connection handler using that ACI pool. Global ACIs are required to allow or deny application administrators to access certain data. These application administrators can then provide more finely-grained access control to users, by placing ACIs directly in the data.

Only the proxy manager can create a pool of ACIs and manage ACIs directly through the ACI data view. Application administrators cannot manage ACIs directly through the ACI data view, even if they have the right to add entries. Application managers can only manage ACIs directly through the data.

ACIs that are defined in the data itself, are evaluated by Directory Proxy Server. These ACIs are entries in the pool of ACIs defined by the proxy manager, that is they are child entries of the entry `cn=`*data-source-name*`,cn=virtual access controls`.

ACIs have a performance impact. Therefore, if you use ACIs within the data itself, keep to a minimum the number of rules in the global ACIs, because these ACIs are evaluated every time the subtree is accessed.

Virtual ACI Syntax

The `dpsaci` attribute resembles the Directory Server `aci` attribute in syntax and behavior. For a description of Directory Server ACI syntax, see "How Directory Server Provides Access Control" on page 42.

The following list describes the differences between virtual ACIs and Directory Server ACIs.

- **Target keywords.** Only the target, targetAttr and targetScope keywords are supported.
- **Permission keywords.** The `All` access write does not permit `selfwrite` operations.
- **Bind rule subject.** For performance reasons, virtual ACIs do not support the `ldap:///suffix??sub?(filter)` as a value for the `userdn` keyword.
- **Bind rule context.** Virtual ACIs do not support SASL authentication. In addition, the `ip` keyword does not support subnet masks.

Virtual ACI Storage and Access

Virtual ACIs are stored centrally, in an LDIF file or in an LDAP directory. When you create a Directory Proxy Server instance, the virtual ACIs are stored in the LDIF file *instance-path* `/config/access_controls.ldif` by default. You can change the location of the virtual ACIs, particularly if you need to share ACIs across multiple proxy servers. For information about how to change the location of virtual ACIs, see "To Define a New ACI Storage Repository" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

The ACI repository is accessed through an LDAP or LDIF data view, depending on the type of repository. By default, the access control data view is an LDIF data view named `virtual access controls`. The view base exposed by the access control data view *must* exist in the ACI repository.

The ACI repository contains one or more pools of ACIs. An ACI pool is defined by an LDAP entry of the type `aciSource`, directly below the view base of the data view. The ACI pool is a subtree of entries. It can contain access controls, and can be the parent entry of other entries containing ACIs.

Virtual ACI Application

Virtual ACIs are applied per connection handler. The name of the ACI pool to be used is defined as the `aci-source` property of the connection handler. Virtual access controls are not evaluated if you bind as the Proxy Manager.

Virtual Schema Checking

Directory Proxy Server exposes its own schema that is different to the schema of a physical data source. The Directory Proxy Server schema can be stored locally in an LDIF file, or in a remote Directory Server. You can configure where the schema is stored with the `dpconf` command. A schema is defined *per connection handler*. The schema for a specific connection handler can be retrieved or updated using `ldapsearch` or `ldapmodify`. When the schema is updated, Directory Proxy Server must be restarted before the changes take effect.

Schema Checking

Generally, schema checking is performed by the server that exposes the schema. In a scenario where Directory Proxy Server acts as a proxy to one or more Directory Servers, the Directory Servers check that add and modify requests adhere to their LDAP schema. When Directory Proxy Server exposes its own schema. Directory Proxy Server must check that add and modify requests adhere to these schema.

Because a schema is defined for a specific connection handler, schema checking is enabled per connection handler. Schema checking is enabled by setting the `schemaCheck` attribute of a connection handler to `true`.

Virtual Data Views and LDAP Groups

With virtual data views, you can define local virtual groups, and use them though ACIs. You can also rely on existing groups defined on backend servers. You can transform the groups from an LDAP directory to appear in the virtual namespace by using DN mapping. You can also transform all member DNs by using attribute value renaming.

With a join data view, you can join two static groups from two different LDAP backends, as long as there are no member naming conflicts. You can also create a read-only virtual group, by using an ACI on the `uniquemember` attribute, for example.

Directory Proxy Server server uses groups in the area of ACIs only. The ACI engine can reference both static and dynamic groups by using the `groupdn` keyword.

Virtual ACIs support both static and dynamic groups. However, the `isMemberOf` feature is not supported. Due to the severe performance impact, nested groups are also not supported.

With dynamic groups, attribute value renaming does not apply to the value of the dynamic group, because this value is an LDAP URL and is therefore not DN syntax. In other words, if a dynamic group value contains a DN, the DN part is not renamed.

CHAPTER 19

Connections Between Directory Proxy Server and Backend LDAP Servers

This chapter describes the connections between Directory Proxy Server and backend LDAP servers. The chapter covers the following topics:

- "LDAP Data Sources" on page 311
- "Connections Between Directory Proxy Server and Backend LDAP Servers" on page 312
- "Forwarding Request From Directory Proxy Server to Backend LDAP Servers" on page 313

LDAP Data Sources

The connections between Directory Proxy Server and backend LDAP servers are configured through *LDAP data sources*. An LDAP data source identifies the name and port numbers of an LDAP server, and the authentication policy that is applied by Directory Proxy Server when forwarding operations to the LDAP server. LDAP data sources also configures how the LDAP server is monitored.

An LDAP data source can be any LDAP v3 server. Certain advanced functionality of Directory Proxy Server might rely on features that are available only in Sun's Directory Server, but the configuration of this functionality is optional. For example, the "Get Effective Rights" control in Sun's Directory Server is used by Directory Proxy Server for proxied authorization.

The health of a backend LDAP server is monitored by testing the connections between Directory Proxy Server and the backend LDAP server. For information about how Directory Proxy Server monitors LDAP data sources, see "How Data Sources Are Monitored" on page 356.

For information about how to create and configure LDAP data sources, see "Creating and Configuring LDAP Data Sources" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Connections Between Directory Proxy Server and Backend LDAP Servers

This section describes how connections between Directory Proxy Server and backend LDAP servers are opened and closed. It also describes the use of connection pools for multiple client requests.

Opening and Closing Connections Between Directory Proxy Server and Backend LDAP Servers

At startup, Directory Proxy Server opens a connection to each data source that is configured, and enabled.

When an error is detected on a connection, Directory Proxy Server closes the connection and tries to reestablish it immediately. If Directory Proxy Server cannot connect to a data source, the data source is considered unavailable. For more information about how Directory Proxy Server responds to failed connections, see “Responding to the Failure of a Data Source” on page 358.

Connection Pools Between Directory Proxy Server and Backend LDAP Servers

Connections between Directory Proxy Server and backend LDAP servers are pooled for use with multiple client requests. Each data source can have one pool of SSL connections and one pool of non-SSL connections. The `ssl-policy` property of the data source and the `is-ssl-mandatory` property of the connection handler determine whether SSL is used when contacting the data source.

The number of connections that can be opened to a data source can be configured independently for BIND, READ, and WRITE operations. The same limit applies to SSL connections and to non-SSL connections.

The following properties can be configured for each data source and for each type of operation:

- The initial number of connections made to the data source
- If more than the initial number of connections are requested, the number of new connections made
- The maximum number of connections that can be made to the data source

When BIND replay is configured, Directory Proxy Server attempts to reuse connections that have already been opened, to optimize performance. If a client opens an authenticated connection, the connection is taken from the BIND pool. Therefore, when BIND replay is used, the connection pool for BIND operations is used more than the connection pools for READ or WRITE operations. For more information about BIND replay, see "Directory Proxy Server Configured for BIND Replay" on page 313.

When a connection to a data source is not used for 5 minutes, the connection is removed from the pool.

Forwarding Request From Directory Proxy Server to Backend LDAP Servers

Client requests can be forwarded from Directory Proxy Server to backend LDAP servers with different levels of authorization and authentication, and with or without the identity of the client. The configuration of the data source determines the way in which a request is forwarded. For information about proxy authorization in client requests, see "Directory Proxy Server Configured for Proxy Authorization" on page 315. For information about how to configure proxy authorization in client requests, see "Proxy Authorization" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

When client requests contain a proxy authorization control, the control is always forwarded with the request, irrespective of how Directory Proxy Server forwards the request. The use case where Directory Proxy Server is configured for proxy authorization *and* the client request itself contains a proxy authorization control is described in "Directory Proxy Server Configured for Proxy Authorization and the Client Request Does Contain a Proxy Authorization" on page 317.

For information about how client requests are forwarded from Directory Proxy Server to backend LDAP servers, see the following sections:

- "Directory Proxy Server Configured for BIND Replay" on page 313
- "Directory Proxy Server Configured for Proxy Authorization" on page 315
- "Directory Proxy Server Configured to Forward Requests As an Alternate User" on page 319
- "Directory Proxy Server Configured to Forward Requests Without the Client Identity" on page 319

Directory Proxy Server Configured for BIND Replay

Directory Proxy Server forwards a BIND request from a client and the credentials of the client to an LDAP server. If the BIND is successful, all subsequent requests from the client to that LDAP server are processed with the authorization of the client.

In BIND replay, if the client makes a subsequent request that is forwarded to another LDAP server, the Directory Proxy Server uses the credentials already provided by the client to BIND to the other LDAP server before forwarding the request.

If a client request contains a proxy authorization control, Directory Proxy Server forwards the control to the backend server.

The following figure shows client identity and credentials being used for authorization by BIND replay.

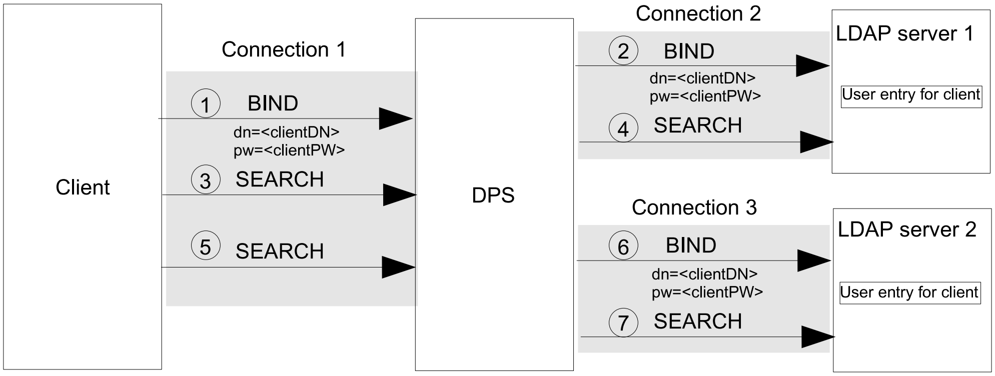


FIGURE 19–1 Authentication in BIND Replay

When Directory Proxy Server is initiated, it opens a connection to each LDAP server. When a client connects to Directory Proxy Server it makes requests in the following stages:

1. The client requests a BIND, and provides a DN and a password.
2. Directory Proxy Server authenticates the client to LDAP server 1 by using the client's credentials. An entry for the client exists in LDAP server 1 and the BIND request is granted.
3. The client issues a SEARCH request that is targeted at LDAP server 1.
4. Directory Proxy Server forwards the SEARCH request to LDAP server 1, reusing connection 2.

   The SEARCH request is performed with the authorization of the client. If the client request contains a proxy authorization control, the request is processed with authorization of the user specified in the proxy authorization control.

   If the client sends more SEARCH requests that are targeted at LDAP server 1, the Directory Proxy Server forwards the request without performing additional binds.
5. The client sends a SEARCH request targeted at LDAP server 2
6. The Directory Proxy Server authenticates the client to LDAP server 2 by using the client's credentials obtained in Step 1. An entry for the client exists in LDAP server 2 and the BIND request is granted.
7. The Directory Proxy Server forwards the SEARCH request to LDAP server 2, reusing connection 3.

If the client is not authenticated to Directory Proxy Server, the BIND request is forwarded as anonymous.

If the client identity is mapped onto another identity, Directory Proxy Server uses the mapped identity to bind to the LDAP server. All requests on that connection are processed with the authorization for the mapped identity. For information about user mapping, see "Directory Proxy Server Configured to Forward Requests As an Alternate User" on page 319.

When Directory Proxy Server is configured for BIND replay, authentication by SASL external bind cannot be used . In BIND replay, Directory Proxy Server authenticates the client to a backend LDAP server by using the client DN and password. In SASL external bind, no password is provided by the client. Furthermore, the password that is stored in the user entry cannot be read in clear text.

For performance reasons, you should configure Directory Proxy Server to use BIND replay only when the extra configuration required for proxy authorization is not feasible, or where proxy authorization is not supported. For information about proxy authorization, see "Directory Proxy Server Configured for Proxy Authorization" on page 315

Directory Proxy Server Configured for Proxy Authorization

When Directory Proxy Server is configured for proxy authorization, Directory Proxy Server can add a proxy authorization control to a client request. The client request is then forwarded with the authorization of the specified in the proxy authorization control.

To simplify the configuration of ACIs, Directory Proxy Server can be configured to allow anonymous reads and to apply proxy authorization for write operations.

If Directory Proxy Server is configured for proxy authorization and the client request contains its own proxy authorization control, Directory Proxy Server does not add a proxy authorization control. In this case, Directory Proxy Server checks with the backend LDAP server that the client has the right to use its proxy authorization control. If the client has the right to use its proxy authorization control, Directory Proxy Server forwards the request with the authorization specified in the client's proxy authorization control.

For information about how to configure proxy authorization in Directory Proxy Server, see "Forwarding Requests With Proxy Authorization" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*

Connections When Directory Proxy Server Is Configured for Proxy Authorization

When Directory Proxy Server is configured for proxy authorization, a client is usually authenticated to the Directory Proxy Server by a non-anonymous BIND or by a SASL external BIND, however, clients can also be anonymous. Directory Proxy Server is usually bound to the data sources by using an administrative identity.

Figure 19–2 shows the connections between a client, Directory Proxy Server, and backend LDAP servers, when Directory Proxy Server is configured for proxy authorization.

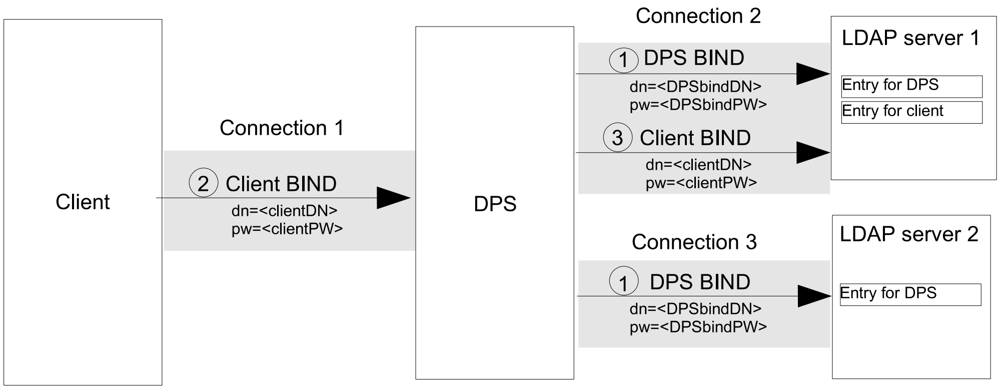


FIGURE 19–2 Connections for Proxy Authorization

The connections for proxy authorization are made in the following stages:

1. When Directory Proxy Server is initiated, it opens a connection to each LDAP server. Directory Proxy Server binds to LDAP server 1 and LDAP server 2 by providing its DN and password, `DPSbindDN` and `DPSbindPW`.

   An entry for `DPSbindDN` exists in both the LDAP servers and the BIND requests are granted. Directory Proxy Server is bound to the LDAP servers, on connection 2 and connection 3.
2. When a client connects to Directory Proxy Server, the client binds by providing its DN and a password, `clientDN` and `clientPW`.
3. The Directory Proxy Server authenticates the client to LDAP server 1 by using the client's credentials and by reusing connection 2.

   An entry for the client exists in LDAP server 1 and the BIND request is granted. The client is bound to Directory Proxy Server on connection 1.

Directory Proxy Server Configured for Proxy Authorization and the Client Request Does Not Contain a Proxy Authorization

Figure 19–3 shows the flow of information when Directory Proxy Server is configured for proxy authorization. The client in Figure 19–2 makes, and Directory Proxy Server adds a proxy authorization control.

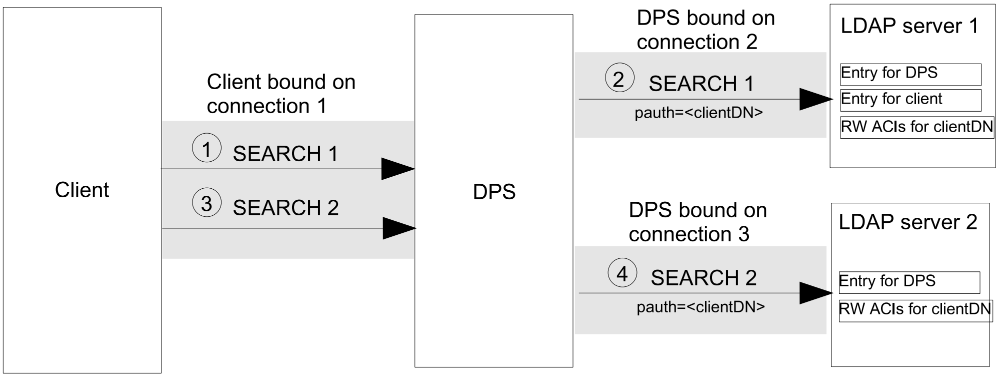


FIGURE 19–3 Information Flow When Proxy Authorization Control Is Added by Directory Proxy Server

1. The client sends a SEARCH request `SEARCH 1`, that does not contain a proxy authorization control. The request is targeted at LDAP server 1.
2. Directory Proxy Server adds a proxy authorization control to the request and forwards the SEARCH operation to LDAP server 1, reusing connection 2.

   The SEARCH operation is performed with the authorization of the user specified in the proxy authorization control. That authorization is defined in the RW ACIs on the LDAP server for the user specified in the proxy authorization control.
3. The client sends a second SEARCH request, `SEARCH 2`, that does not contain a proxy authorization control. The request is targeted at LDAP server 2.
4. The Directory Proxy Server forwards the SEARCH operation to LDAP server 2, reusing connection 3.

   Notice that it is not necessary for the client to bind to LDAP server 2 before the request can be processed, and it is not necessary for the LDAP server to contain an entry for the client.

Directory Proxy Server Configured for Proxy Authorization and the Client Request Does Contain a Proxy Authorization

Figure 19–3 shows the flow of information when the client in Figure 19–2 makes a request that *does* contain a proxy authorization control. Directory Proxy Server verifies that the client has the right to use its proxy authorization control.

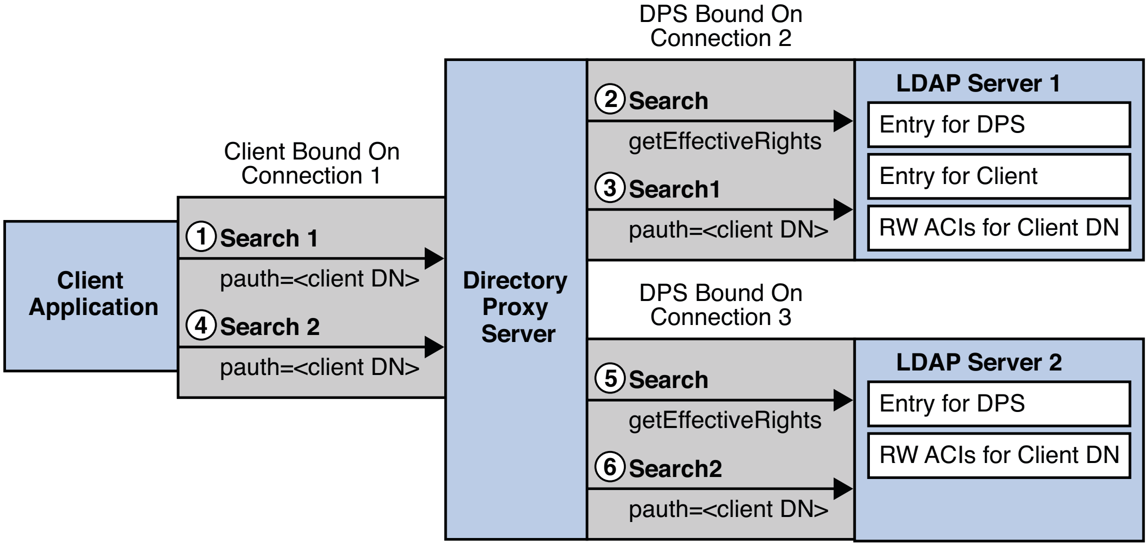


FIGURE 19–4 Information Flow When Proxy Authorization Control Is Contained in the Client Request

1. The client sends a SEARCH request `SEARCH 1`, that contains a proxy authorization control. The request is targeted at LDAP server 1.
2. Directory Proxy Server verifies that the `clientDN` has the right to use a proxy authorization control on LDAP server 1, by getting the effective rights of the client on LDAP server 1. For information about how to get effective rights, see "Viewing Effective Rights" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*
3. Directory Proxy Server forwards the SEARCH operation to LDAP server 1, reusing connection 2.

   The SEARCH operation is performed with the authorization of the user specified in the proxy authorization control. The authorization is defined in the RW ACIs on the LDAP server.
4. The client sends a second SEARCH request, `SEARCH 2`, that contains a proxy authorization control. The request is targeted at LDAP server 2.
5. Directory Proxy Server verifies that the `clientDN` has the right to use a proxy authorization control on LDAP server 2, by getting the effective rights of the client on LDAP server 2.
6. The Directory Proxy Server forwards the SEARCH operation to LDAP server 2, reusing connection 3.

   Notice that it is not necessary for the client to bind to LDAP server 2 before the request is processed, and it is not necessary for the LDAP server to contain an entry for the client.

Security Issues When Directory Proxy Server Is Configured for Proxy Authorization

Consider the following security risks before configuring Directory Proxy Server for proxy authorization:

- When Directory Proxy Server is configured for proxy authorization, it assumes the rights of any client for which it forwards a request. A Directory Proxy Server that is not authorized to perform write operations on data, can perform those operations by using proxy authorization.
- An LDAP server must contain an entry with the appropriate R/W ACIs for the user specified in the proxy authorization control. If the entry was accessed illegally by a third party, that party might be able to impersonate.
- The authorization identity configured in the proxy authorization control must be protected from tampering.

Directory Proxy Server Configured to Forward Requests Without the Client Identity

In some deployment scenarios, it is not necessary to maintain the identity of a client when the client makes request. Directory Proxy Server can be configured to forward requests to LDAP servers without the client identity. The LDAP servers process the requests with the identity and authorization of the Directory Proxy Server.

Directory Proxy Server Configured to Forward Requests As an Alternate User

Client requests can be performed with the identity of an alternate user by using the feature called *user mapping*. In user mapping, the client identity is mapped to the identity of an alternate user. After a BIND operation, the Directory Proxy Server submits subsequent operations as the alternate user.

When a client identity is mapped to another identity, requests from that client can be forwarded to the backend LDAP servers by using BIND replay or by using proxy authorization.

Client identities can be mapped to alternate identities either locally on the Directory Proxy Server or remotely on an LDAP server. Figure 19–5 and Figure 19–6 illustrate local mapping and remote mapping.

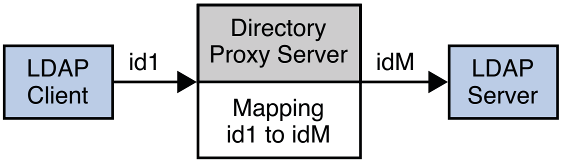


FIGURE 19–5 Local Mapping of a Client Identity to an Alternate Identity

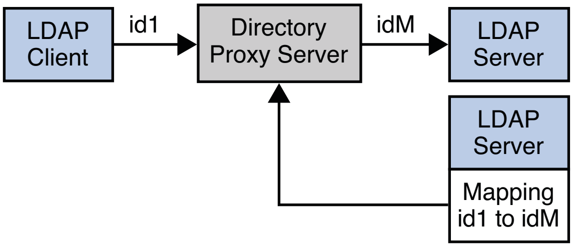


FIGURE 19–6 Remote Mapping of Client Identity to an Alternate Identity

In local mapping, the identity mapping is configured in the Directory Proxy Server. The configuration cannot be changed without reconfiguring the Directory Proxy Server. Local mapping can be configured for unauthenticated clients, authenticated clients, and for clients authenticated by proxy.

In remote mapping, the identity mapping is configured in an entry in the remote LDAP server. The mapping can be changed by modifying the entry in the remote LDAP server. It is not necessary to reconfigure the Directory Proxy Server to change the mapping. Remote mapping can be configured for unauthenticated clients and for clients authenticated by proxy.

Remote mapping must not be used for data sources configured for BIND replay. In BIND replay, the Directory Proxy Server forwards a client request by using the authentication provided in the BIND operation. However, in remote mapping the client DN and password provided in the BIND operation are mapped to an alternate DN and password. The client's password cannot be retrieved from the backend LDAP sever.

If the user mapping is enabled but the mapping fails, the client identity is mapped to a default identity. A user mapping can fail when a client identity is mapped to a non-existent alternative identity or when there has been a configuration error.

For information about how to configure user mapping, see “Forwarding Requests as an Alternate User” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*

CHAPTER 20

Connections Between Clients and Directory Proxy Server

All the incoming connections to Directory Proxy Server are categorized into connection handlers according to a set of criteria. A connection handler defines the resource limits and request filters that apply to the connection, and the data views that are exposed to the connection.

This chapter covers the following topics:

Criteria for Allocating a Connection to a Connection Handler

An instance of Directory Proxy Server can have many connection handlers. When a client connects to Directory Proxy Server, the proxy evaluates whether the attributes of the connection match the criteria of one of the connection handlers. When a match is found, the connection is classified into that connection handler. All of the policies defined for that connection handler apply to the connection. Operations performed through that connection are exposed to all of the data views or to a list of data views defined by the connection handler.

After being classified into a connection handler, a connection can be automatically reclassified into another connection handler by Directory Proxy Server. For example, if a client connects anonymously, the connection is allocated to the connection handler configured for anonymous connections. If the client later provides a bind DN on the same connection, the connection can be reallocated to another connection handler. Similarly, a non-secure LDAP connection is initially classified into a connection handler for non-secure connections. If the client uses startTLS to promote the connection to secure mode, the connection is automatically reclassified into a connection handler for secure connections.

A connection is evaluated against connection handlers in order of the priority of the connection handler. Priority one is the highest priority connection handler. The connection is classified into the first connection handler for which there is a match. Connection handlers with the most specific criteria should have a higher priority than those with less specific or more general criteria. For example, a connection handler that specifies a bind DN should have a higher priority than a connection handler that specifies a simple bind.

If a connection does not match the criteria of any configured connection handler, the connection is allocated to the *default connection handler*. The criteria of the default connection handler cannot be modified. In addition, the default connection handler cannot be disabled or deleted. However, the policies and data views of the default connection handler can be changed.

The default connection handler is the lowest priority connection handler. If a new connection handler is created without a priority, the new connection handler is given a higher priority than the default connection handler. If two connection handlers have the same priority, the order in which the connection is evaluated against them is not specified.

The criteria expression of a connection handler is a logical `AND` between criteria of different types and a logical `OR` between criteria of the same type. For example, if a criteria is specified for client IP address and a criteria is set for client domain name, both of the criteria must be met. However, if two criteria are set for client IP address, either, not both, of the criteria must be met.

The following list summarizes the criteria used to classify connections into connection handlers. For information about how to configure the criteria, see “Creating, Configuring, and Deleting Connection Handlers” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

- **Client IP address and mask.** A set of IPv4 or IPv6 address masks. The IP address of a client connection must match at least one of the masks in order for the connection to be accepted by the connection handler.

  The IP address can be in one of the following formats:

  - IP address in dotted decimal form. For example, `129.153.129.14`.
  - IP address and bits, in the form of `network number/mask bits`. For example, `129.153.129.0/24`.
  - IP address and quad, in the form of a pair of dotted-decimal quads. For example, `129.153.129.0/255.255.255.128`.
  - All addresses:`ALL`, a catch-all for clients that are not placed into other, higher priority, groups.
  - `0.0.0.0`. This address is for groups for which initial membership is not considered. For example, for groups that clients switch to after their initial bind.
  - IP address of the local host. IP address `127.0.0.1` is the IP address of a client that is running on the same machine as Directory Proxy Server.

- **Client domain name.** A set of domain names. A client network domain must match at least one of the suffixes in order for the connection to be accepted by the connection handler.

  In order to be able to filter the client's domain name, Directory Proxy Server must be able to convert the incoming IP address into the fully qualified domain name. If the naming service returns a hostname without the domain name, Directory Proxy Server cannot filter the client's domain name.

  Directory Proxy Server does not assume any domain suffix, therefore the fully qualified domain name must be provided. A domain name suffix with a leading period, for example, `.sun.com`, will cause all hosts with domain names that end in that suffix to match.

  The domain name can be in one of the following formats:

  - Full name, for example, `box.eng.sun.com`.
  - Suffix name, for example, `.eng.sun.com`. If the suffix name is used to identify clients, ensure that DNS is set up to return fully qualified names to the DNS queries.
  - Fully qualified name of the local host. This criteria is for a client that is running on the same machine as Directory Proxy Server.
- **Bind DN.** A regular expression that must be matched by the bind DN of a client.

  For example, the following regular expression could be used as a bind DN criteria for a connection handler: `uid=(.*),dc=example,dc=com`. A client that binds with a `uid` such as `uid=user1,dc=example,dc=com` matches the criteria and can be allocated to the connection handler. A client that binds with another DN such as `ou=accounts,dc=example,dc=com` does not match the criteria and cannot be allocated to the connection handler.
- **LDAP search filter.** A search filter that the entry of a bound client must match.

  For example, the following filter could be used as a criteria for a connection handler: `uid>=1000`. Bound clients with a `uid` that matches the filter can be allocated to the connection handler.
- **Authentication method.** An authentication method that must match the client entry in order for the connection to be accepted by the connection handler.

  The authentication method can be one of the following:

  - SIMPLE
  - SASL/EXTERNAL
  - Anonymous
- **IP port.** A set of IP port numbers. A client connection must come through one of the specified ports in order for the connection to be accepted by the connection handler.
- **SSL connection.** A flag indicating whether or not client connections must use SSL in order to be accepted by the connection handler.

Data Views for Connection Handlers

When a connection is allocated to a connection handler, requests on the connection are exposed to a list of data views configured for that connection handler. The list of data views for a connection handler can contain zero, one, or multiple data views.

If the list of data views is empty, requests on the connection are not distributed to any data view. Applications using the connection cannot access any data and a `No such Object` error is returned.

If the list of data views contains multiple data views, requests on the connection are distributed to the data view that most specifically corresponds to the target DN of the request. For example, in Figure 20–1, requests on a connection in `connection-handler-1` can be distributed to `data-view-2`, `data-view-3` or `data-view-4`. However, if a search request has a target DN of `ou=people,dc=example,dc=com`, the request is distributed either to `data-view-3` or to `data-view-4`.

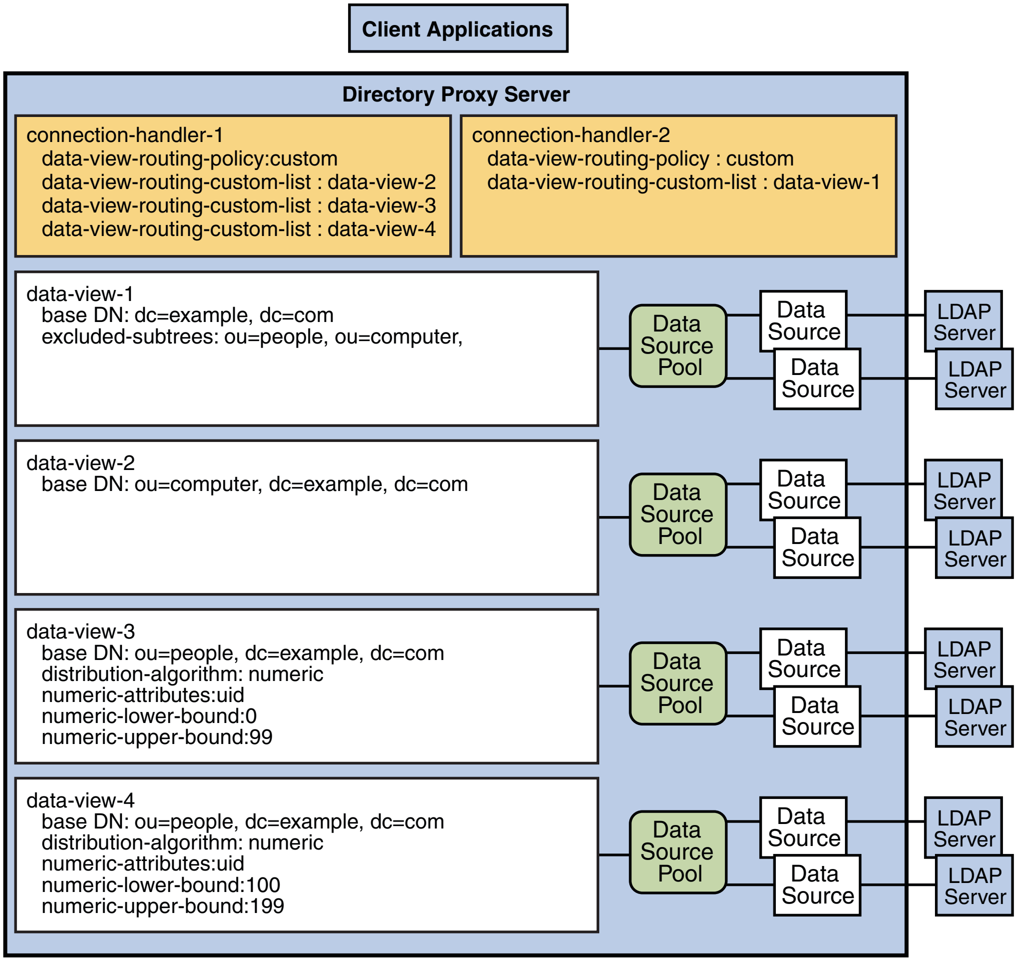

Client Applications
Directory Proxy Server
connection-handler-1
data-view-routing-policy:custom
data-view-routing-custom-list : data-view-2
data-view-routing-custom-list : data-view-3
data-view-routing-custom-list : data-view-4
connection-handler-2
data-view-routing-policy : custom
data-view-routing-custom-list : data-view-1
data-view-1
base DN: dc=example, dc=com
excluded-subtrees: ou=people, ou=computer,
data-view-2
base DN: ou=computer, dc=example, dc=com
data-view-3
base DN: ou=people, dc=example, dc=com
distribution-algorithm: numeric
numeric-attributes:uid
numeric-lower-bound:0
numeric-upper-bound:99
data-view-4
base DN: ou=people, dc=example, dc=com
distribution-algorithm: numeric
numeric-attributes:uid
numeric-lower-bound:100
numeric-upper-bound:199
Data Source Pool
Data Source
Data Source
LDAP Server
LDAP Server

FIGURE 20–1 List of Data Views in a Connection Handler

Affinity can be defined between a client connection and the data view selected to respond to requests on that connection. This feature is called *data view affinity*. When data view affinity is enabled, successive requests on a client connection are exposed exclusively to the data view used for the first request on that connection.

When data view affinity is enabled it takes precedence over other types of routing. For example, in Figure 20–1, a search request with a target DN of `ou=computer,dc=example,dc=com` is exposed to `data-view-2`. All subsequent requests on that client connection are exposed exclusively to `data-view-2`. If a subsequent request on that client connection has a target DN of `ou=people,dc=example,dc=com`, the request is exposed to the data view for `ou=computer,dc=example,dc=com`, not the data view for `ou=people,dc=example,dc=com`.

For information about how to configure data view affinity, see “To Configure Affinity for Data Views” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Resource Limits Policies for Connection Handlers

A resource limits policy defines the maximum resources that Directory Proxy Server can process for a given connection handler. By using this type of connection handler policy, you can limit the resources allocated to connections, requests, and referrals.

A connection handler can have zero or one resource limits policy. If no resource limits policy is defined, no limits are applied to connections, requests and referrals. For information about how to configure resource limits policies and examples of resource limits policies, see “Creating and Configuring a Resource Limits Policy” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

The following list summarizes the resource limits that can be configured:

- Connections
  - Maximum number of connections.
  - Maximum number of simultaneous connections from a single client.
  - Maximum number of operations per connection. If a client exceeds the maximum number of operations on one connection, the connection is closed by Directory Proxy Server.
  - Maximum number of simultaneous operations per connection.

    If the maximum number of simultaneous operations per connection is `1`, clients must perform synchronous operations. Additional requests for simultaneous operations, except for requests to abandon an operation, will fail with a `Server Busy` error.
- Searches
  - Maximum permitted size of a search operation result
  - Maximum permitted duration of a search operation
  - Minimum length of a substring allowed in a search filter
  - Customized search limits, described in “Customized Search Limits” on page 327
- Referrals
  - Maximum number of hops when following referrals
  - Bind policy to be applied when referrals are followed:
    - Use password if supplied, else follow the referral as anonymous
    - Always follow the referral as anonymous
  - Policy applied when a referral is returned by the server:
    - Follow referrals
    - Forward referrals to the client

- Discard referrals

For information about how to configure a resource limits policy, see "Creating and Configuring a Resource Limits Policy" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*

Customized Search Limits

Customized limits can be defined for search operations, based on the search base and search scope. If the target DN of a search is specified in a list, and the scope of a search is one-level or subtree, the maximum size of the search result can be configured.

Custom search limits are defined for a specific resource limits policy. If the resource limits policy is deleted, the custom search limits defined for that policy are also deleted. If custom search limits are not specified, standard search size limits are applied.

Request Filtering Policies for Connection Handlers

Request filtering policies control access of clients to data. A connection handler can reference zero or one request filtering policy.

The following aspects of client access can be defined by using this type of connection handler policy:

- The types of operation that clients are allowed to perform or are prohibited from performing.

  Each of the following types of operation can be allowed or prohibited: add, bind, compare, delete, extended operations, modify, modify DN, search, and search based on inequality filters.
- Attributes that are allowed or prohibited from being used in search filters and compare operations.

  All attributes can be permitted in search filters and compare operations, or a list of attributes can be permitted or prohibited.
- The scope of search operations.

  The scope can be the base DN, one level below the base DN, or the entire subtree below the base DN.
- The subtrees that clients are allowed to access or are prohibited from accessing.

  For information, see "Subtrees in the Request Filtering Policy" on page 328.
- Entries that can be accessed in search operations and data that can be returned by search operations.

  For information, see "Search Data Hiding Rules in the Request Filtering Policy" on page 328.

For information about how to configure a request filtering policy, see "Creating and Configuring Request Filtering Policies and Search Data Hiding Rules" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Subtrees in the Request Filtering Policy

The request filtering policy is configured with one or more allowed subtrees and zero, one, or more prohibited subtrees. The subtrees identify the part of a data view that can be accessed by clients.

Allowed Subtrees

An allowed subtree is specified by a minimum base DN. Clients are permitted to perform operations on entries at the minimum base DN or below the minimum base DN. By default, the minimum base DN is the root DN.

If a client requests a search operation that is targeted at a DN superior to the minimum base DN, Directory Proxy Server rewrites the DN to target the minimum base DN. If a client performs any other operation that is targeted at a DN superior to the minimum base DN, the operation is denied.

Prohibited Subtrees

A prohibited subtree is a branch of the allowed subtree that cannot be accessed by the client. The base DN of a prohibited subtree must be subordinate to the minimum base DN of an allowed subtree. If a client performs an operation that is targeted at a prohibited subtree, the operation is denied.

Search Data Hiding Rules in the Request Filtering Policy

Rules that determine how to return the result of a search operation to a client are called *search data hiding rules*. For information about creating search data hiding rules, see "To Create Search Data Hiding Rules" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

The result of a search operation can be returned in one of the following ways:

- The target entry is not returned
- The target entry is returned but the specified attributes are filtered out
- The target entry is returned but the unspecified attributes are filtered out

Search data hiding rules can be applied to the following entries:

- Entries with the specified DN
- Entries with the specified DN pattern
- Entries with a specified attribute name/attribute value pair (*attrName*:*attrValue*)

Search data hiding rules are defined for a given request filtering policy and cannot be used by another request filtering policy. If a request filtering policy is deleted, its associated search data hiding rules are automatically deleted. Zero, one or multiple search data hiding rules can be defined in one request filtering policy.

CHAPTER 21

Directory Proxy Server Client Authentication

This chapter describes how Directory Proxy Server identifies incoming client connections. The chapter covers the following topics:

Client Authentication Overview

Client authentication determines how a client identifies itself to Directory Proxy Server.

From a protocol perspective, client authentication can occur at two levels:

- **LDAP level.** Authentication occurs in the LDAP bind operation.
- **Connection level.** Authentication occurs in the network connection established between the client and Directory Proxy Server.

Directory Proxy Server can also be configured to accept client requests without authentication.

The following list summarizes the supported authentication options. These options are discussed in more detail in the remainder of this chapter.

- **Simple bind authentication.** Simple bind authentication occurs at the bind level. When the client binds, it provides a unique name (bind DN) and password to Directory Proxy Server. Directory Proxy Server forwards these credentials, along with the bind request, to a backend LDAP server.

  Simple bind authentication can also be made over a secure connection. However, the server still identifies the client from its bind DN.
- **Certificate-based authentication** Certificate-based authentication occurs at the connection level when the connection is secure. When authentication occurs at the connection level, the client connects over an encrypted (SSL) connection and provides a certificate. Directory Proxy Server checks the validity of the client certificate and maps the certificate to an LDAP DN.
- **No authentication.** If the client does not provide a certificate, or a bind DN and password, no authentication occurs. In this case, the client connects to Directory Proxy Server anonymously. This is known as *anonymous access*.

Simple Bind Authentication

Simple bind authentication is the most common way to authenticate LDAP clients. In a simple bind, the client either binds anonymously, that is, with an empty bind DN, or by providing a DN and a password. Directory Proxy Server binds to a data source to validate the credentials and to authenticate the client. An entry for the client must exist on the data source, otherwise the client is considered to be anonymous. When a client is authenticated, Directory Proxy Server records the identity of the client.

Directory Proxy Server is configured for simple bind authentication by default. No additional configuration is required. Because the client provides a password to Directory Proxy Server, simple bind authentication is also known as *password-based authentication*.

Password Encryption and Verification

The way in which passwords are encrypted and checked depends on the type of *data view* through which the client accesses the data source. For information about data views, see Chapter 17, “Directory Proxy Server Distribution,” and Chapter 18, “Directory Proxy Server Virtualization.”

For LDAP data views, Directory Proxy Server relies on the backend LDAP server for password encryption and verification. When a client modifies a password by using an ADD or MODIFY operation, the backend LDAP server can apply a password encryption policy when it stores the password. When the client issues a BIND request, the backend LDAP server is responsible for verifying the password.

For LDIF and JDBC data views, Directory Proxy Server is responsible for password encryption and verification.

- LDIF data views. When a client modifies a password, Directory Proxy Server applies the encryption policy defined by the `db-pwd-encryption` property of the data view. The encryption policy can be `PLAIN`, `SHA`, or `SSHA`. The password is still stored in the data source, that is, in the LDIF file.
- JDBC data views. When a client modifies a password, Directory Proxy Server applies the 3DES encryption mechanism to encrypt the JDBC data source password.

When encrypted passwords are stored, the encrypted value is prefixed by the encryption policy. So for example, a stored, encrypted password might look like `{SSHA}mcasopjebjakiue` or `{SHA}askjdlaijfbnja`. When the client issues a BIND request, Directory Proxy Server verifies the password and expects the encryption policy tag.

Certificate-Based Authentication

Certificate-based authentication over an SSL connection is the most secure type of authentication. Therefore, when authentication occurs at the connection layer, the client does not need to provide an additional name (bind DN) and password to Directory Proxy Server during the LDAP bind.

A client can only perform certificate-based authentication over an SSL connection. The basic steps in establishing an SSL connection are as follows:

1. The client requests that a secure connection be established.

   As part of this request, Directory Proxy Server provides a *server certificate* to the client. A server certificate is a single certificate associated with one instance of Directory Proxy Server. When a secure connection is used, the server certificate identifies the instance of Directory Proxy Server to the client.

   The establishment of the connection includes a negotiation phase. During this phase, the client and Directory Proxy Server attempt to agree on the encryption policy that is used. The server certificate contains the list of encryption policies (ciphers) that are supported by the Directory Proxy Server.
2. Depending on the security configuration of the proxy server, the server might require the client to provide a certificate.
3. The client provides a certificate to the server, either because the client is configured to do so, or because the proxy server has requested it.
4. The client then sends an LDAP bind request to Directory Proxy Server to establish the client's identity on that connection.
5. If the request is a simple bind, Directory Proxy Server uses the bind DN and password provided by the client.

6. If the request is a SASL external bind, Directory Proxy Server does one of two things:
   - Considers the subject of the certificate as the bind DN of the client.
   - Maps the certificate by searching the backend server for an entry that matches the received certificate. If the `verify-certificates` property is set, Directory Proxy Server verifies that the received certificate is the one stored in the entry that is found.

     The following configuration properties determine how Directory Proxy Server performs that search:

     ```
     cert-data-view-routing-policy
     cert-data-view-routing-custom-list
     cert-search-bind-dn
     cert-search-bind-pwd-file
     cert-search-base-dn
     cert-search-attr-mappings
     ```
7. When the proxy server has the bind DN, it can verify the validity of the client.

For more information about SSL for Directory Proxy Server, see “Secure Sockets Layer for Directory Proxy Server” on page 340.

For certificate-based authentication to occur, Directory Proxy Server must be configured to accept client certificates and the client must be configured to use SASL external bind.

Configuring Certificates in Directory Proxy Server

When you create a Directory Proxy Server instance, the certificate database is automatically populated with the CA certificates of certain trusted CAs. You can add trusted CA certificates to the certificate database if necessary, by using the Directory Service Control Center (DSCC) or by using the `dpadm` command. For more information, see “To Install a CA-Signed Server Certificate for Directory Proxy Server” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

When a client provides a certificate to Directory Proxy Server, the server verifies that certificate against the list of trusted CA certificates in its certificate database. The verification is successful if the server's certificate database contains the client certificate itself, or the CA certificate with which the client certificate was generated.

The server certificate can be one of the following:

- **Self-signed certificate.** A public and private key pair, where the public key is signed by Directory Proxy Server.
- **Trusted CA certificate.** A single certificate that is automatically generated by the company’s internal certificate server or by a known Certificate Authority (CA).

Directory Proxy Server also supports the use of a *server certificate chain*. A server certificate chain is a collection of certificates that are automatically generated by the company's internal certificate server or by a known CA. The certificates in a chain trace back to the original CA, providing proof of identity. This proof is required each time you obtain or install a new server certificate.

When an instance of Directory Proxy Server is created, a default self-signed certificate is created. By default, Directory Proxy Server manages the SSL certificate database password internally.

You can install any number of certificates on a server. When you configure SSL for an instance of Directory Proxy Server, you must install at least one server certificate and one trusted CA certificate.

For an explanation of how certificate-based authentication works, see "Certificate-Based Authentication" on page 333. For information about how to configure certificate-based authentication for Directory Proxy Server, see "To Configure Certificate-based Authentication" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Using SASL External Bind

When a client binds to Directory Proxy Server with the Simple Authentication and Security Layer (SASL) external bind, Directory Proxy Server obtains the credentials of the client from the certificate, rather than from the bind DN.

The server obtains the credentials in one of two ways:

- Considers the subject of the certificate as the bind DN of the client
- Maps the certificate subject to data within its own database, to deduce the bind DN

SASL external bind cannot be used if Directory Proxy Server is configured for BIND replay. In BIND replay, Directory Proxy Server authenticates the client to a backend LDAP server by using the client DN and password. In SASL external bind, no password is provided by the client. Furthermore, the password that is stored in the user entry cannot be read in clear text. For information about bind replay, see "Directory Proxy Server Configured for BIND Replay" on page 313.

SSL can be used to protect subsequent interactions between the client and Directory Proxy Server.

For information about how to configure authentication by SASL external bind, see "To Configure Directory Proxy Server for SASL External Bind" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Anonymous Access

Anonymous access makes data available to any client, regardless of whether the user has authenticated.

For information about how to configure Directory Proxy Server for anonymous connections from clients, see “To Configure Anonymous Access” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Directory Proxy Server Client Listeners

Directory Proxy Server enables you to configure certain aspects of a client connection through a *client listener*. Two client listeners are provided, a secure listener (`ldaps-listener`) and a non-secure listener (`ldap-listener`).

The secure listener specifies that the connection is made to a secure port, over SSL. The non-secure listener specifies that the connection is made to a non-secure port, without SSL. Clients use either the secure listener or the non-secure listener, depending on the type of connection required by that client.

Note – A client can also establish a secure connection to a non-secure port if the client supports Start TLS.

Both the secure and non-secure listener specify the following aspects of a client connection:

`is-enabled`	Specifies whether clients are able to use that listener to connect to Directory Proxy Server
`listen-port`	The port number on which Directory Proxy Server listens for client connections
`listen-address`	The IP address of the listener
`connection-idle-timeout`	The maximum time a client connection can remain idle before being closed
`connection-read-data-timeout`	The maximum time that a listener can wait for new data to be available
`connection-write-data-timeout`	The maximum time that a listener can wait to send results back to clients
`max-connection-queue-size`	The maximum size of a listener's connection queue
`max-ldap-message-size`	The maximum size of an LDAP message.

`number-of-threads` The number of threads allocated to a listener to for simultaneous client connections and requests

`use-tcp-no-delay` Whether or not `TCP_NODELAY` is enabled for connections between a client and Directory Proxy Server

For information about how to configure listeners, see "Configuring Listeners Between Clients and Directory Proxy Server" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

CHAPTER 22

Security in Directory Proxy Server

This chapter describes the mechanisms that can be used to secure data that passes through Directory Proxy Server.

The chapter covers the following topics:

- "How Directory Proxy Server Provides Security" on page 339
- "Secure Sockets Layer for Directory Proxy Server" on page 340
- "Ciphers and Protocols for Directory Proxy Server" on page 341

How Directory Proxy Server Provides Security

Directory Proxy Server provides security through a combination of the following methods:

- Encryption

  Encryption protects the privacy of information. When data is encrypted, the data is scrambled in a way that only a legitimate recipient can decode. Directory Proxy Server supports SSL encryption. For information about SSL, see "Secure Sockets Layer for Directory Proxy Server" on page 340.
- Authentication

  Authentication is a means for one party to verify another's identity. For example, a client gives a password to Directory Proxy Server during an LDAP bind operation. Policies define the criteria that a password must satisfy to be considered valid, for example, age, length, and syntax. Directory Proxy Server supports anonymous authentication, password-based authentication, and certificate-based authentication. For information about authentication, see Chapter 21, "Directory Proxy Server Client Authentication."
- Access control instructions (ACIs)

  ACIs govern the access rights granted to client applications, and provide a way of specifying required credentials or bind attributes. Directory Proxy Server implements access control through request filtering policies and through virtual ACIs. For information about request filtering policies , see "Request Filtering Policies for Connection Handlers" on page 327. For information about virtual ACIs, see "Access Control On Virtual Data Views" on page 306.

- Auditing and Logs

  Auditing can be used to determine whether security has been compromised. The log files maintained by Directory Proxy Server can be audited to track who has accessed the server, and what operations they have performed. For information about log files, see Chapter 24, "Directory Proxy Server Alerts and Monitoring," and Chapter 23, "Directory Proxy Server Logging."

Secure Sockets Layer for Directory Proxy Server

The Secure Sockets Layer (SSL) provides encrypted communications between a client and Directory Proxy Server. By using SSL with authentication, data sent to and from Directory Proxy Server can be encrypted.

When an instance of Directory Proxy Server is created, SSL is enabled by default and the following directories and files are created:

A randomly generated password to protect the certificate database
    The password is stored in *instance-path*`/etc/pass.txt`

A key store database for certificates
    The keystore database is located in *instance-path*`/alias/cert.jks`

A key store database for a symmetric encryption key
    The keystore database is located in *instance-path*`/alias/key.jceks`

The key store databases are protected by the same password.

For more information about SSL, see "Secure Sockets Layer (SSL)" on page 88. For information about how to configure SSL between clients and Directory Proxy Server, see "Configuring Listeners Between Clients and Directory Proxy Server" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*

Directory Proxy Server supports the Start TLS extended operation. StartTLS can be used to provide security over a regular LDAP connection. With StartTLS, clients can bind to a non-secure port and then use the TLS protocol to initiate a secure connection.

Ciphers and Protocols for Directory Proxy Server

The ciphers and protocols that can be used by Directory Proxy Server depend on the JVM that is used. By default, Directory Proxy Server uses the default ciphers and protocols for the JVM.

You can retrieve a list of ciphers and protocols by using the `dpconf` command:

Enabled ciphers
The list of ciphers that are currently enabled for both the LDAP and LDAPS listeners. Because the LDAP and LDAPS listeners are synchronized, the properties are part of the global server configuration, and not the listener configuration.

Supported ciphers
The list of ciphers supported by the JVM for Directory Proxy Server.

Enabled protocols
The list of protocols that are currently enabled for both the LDAP and LDAPS listeners. Because the LDAP and LDAPS listeners are synchronized, the properties are part of the global server configuration, and not the listener configuration.

Supported protocols
The list of protocols supported by the JVM for Directory Proxy Server.

For reference information about cipher suites, see “Cryptographic Algorithms Used With SSL” on page 90. For information about how to choose ciphers, see “Choosing SSL Ciphers and SSL Protocols for Directory Proxy Server” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

CHAPTER 23

Directory Proxy Server Logging

Directory Proxy Server logs information in access logs and error logs. Additionally, a plug-in can be configured to log messages to a syslog daemon. Unlike Directory Server, Directory Proxy Server does not provide an audit log.

Log files for Directory Proxy Server can be configured through Directory Service Control Center or on the command line. For information about how to configure log files, see Chapter 28, "Directory Proxy Server Logging," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

For information about access logs and error logs, see the following sections:

- "Introduction to Directory Proxy Server Logs" on page 343
- "Log File Rotation" on page 344
- "Log File Deletion" on page 344
- "Message Severity" on page 345
- "Error Logs for Directory Proxy Server" on page 345
- "Access Logs for Directory Proxy Server" on page 348
- "Tracking Client Requests Through Directory Proxy Server and Directory Server Access Logs" on page 351

Note that the log message format is still evolving in this release of Directory Proxy Server.

Introduction to Directory Proxy Server Logs

The Directory Proxy Server logging service provides access logs and error logs. The logs are flat files that contain information about client operations and about the health of Directory Proxy Server. By default, log files are stored under *instance-path*`/logs` with the permission of `600`. If an instance of Directory Proxy Server is started without valid log files, log files are created in the default location and a warning is sent to DSCC.

You can configure the following aspects of the logs:

- Set the log level for each message category
- Globally set the default log-level for all message categories
- Globally enable all logs
- Set the name, location and permissions of log files
- Set the maximum number of log files
- Define a rotation policy for each log file
- Include or exclude search filters in access log messages for search operations

Log messages can also be sent to the syslog daemon. For information about how to log messages to a syslog daemon, see "Logging Alerts to the syslogd Daemon" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Log File Rotation

Log files can be rotated manually at any time, or can be rotated automatically when the following events occur:

- When the log reaches a specified size
- At a specified start-time, start-day, and interval
- At a specified start-time, start-day, and interval, if the log file is bigger than a specified size

The start-time, start-day, and interval can have the following combinations:

- Time-of-day followed by an interval of days, hours, or minutes
- Day-of-week and time-of-day, followed by an interval of weeks
- Day-of-month and time-of-day, followed by an interval of months

The time-of-day takes precedence over the interval. For example, a log that is specified to be rotated at 3am and then every 10 hours is rotated at the following times: 03:00, 13:00, 23:00, and again at 03:00 (not 07:00).

If the log is configured for rotation on the 31st of the month but the month has fewer than 31 days, the log is rotated on the first day of the following month.

Log File Deletion

A log file deletion policy defines when backup log files are deleted. The log file currently in use is never deleted by a deletion policy.

The following deletion policies can be enabled:

- **Deletion based on time.** Log files are deleted when they reach a specified age.
- **Deletion based on size.** Log files are deleted when the total size of all the log files reaches a specified limit. The size of the current log file is taken into account, although this file is not deleted.
- **Deletion based on free disk space.** When the free disk space reaches a specified minimum, the oldest backup log file is deleted. If the free disk space is still lower than the minimum, the next oldest backup log file is deleted, and so forth.

By default, log file deletion is based on free disk space, with a default value of 1 Megabyte. When all three deletion policies are activated simultaneously, they are processed in order of time, size, and free disk space. For information about how to configure log file deletion, see “Deleting Directory Proxy Server Logs” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Message Severity

Messages are included in log files or filtered out of log files according to the severity of the message, the category of the message, and the log-level that has been configured for that category. The categories and log-levels for the error logs and access logs are different, and are discussed in the sections that follow.

Messages are ranked according to their severity. Messages can have one of the following severities, where `error` is highest severity and `debug` is the lowest severity:

1. `error`
2. `warning`
3. `info`
4. `debug`

Messages with a severity that is lower than the log-level configured for its message category are not included in the log file. Messages with a severity that is equal to or higher than the log-level configured for its associated message category are included in the log file.

Error Logs for Directory Proxy Server

Error logs contain information about the health of the Directory Proxy Server. Error messages are categorized according to the cause of the message. The following table lists the categories of messages that can be included in an error log.

TABLE 23–1 Message Categories for Error Logs

Category Name	Category Description
`CONFIG`	Information about configuration
`DECODE`	Information about operation decoding
`PLUGIN`	Information about plug-in processing
`PROCESSING`	Information about a significant event that occurred during client processing
`BACKEND`	Information about an operation with a data source
`INTERNAL`	Information about an internal error in the core server
`SHUTDOWN`	Information about an event at server shutdown
`STARTUP`	Information about an event at server startup

Error Log Levels

Each message category can be configured with one of the following log-levels:

1. `none` No messages are included in the log file.
2. `error` Only error messages are included in the log file.
3. `warning` Error messages and warning messages are included in the log file.
4. `info` Errors, warnings and informational messages are included in the log file.
5. `all` All messages are included in the log file. In most cases, this setting produces the same results as the `info` setting. In certain situations, this setting enables additional debugging messages to be logged.
6. `inherited` The log level is inherited from the value of the `default-log-level` property.

By default, the log level for each message category is `info`.

The log-level of a message category works in conjunction with the severity level of a message to determine which messages are included in the log file. For more information, see “Message Severity” on page 345.

Format of an Error Message

An error log message has this format:

timestamp - *message category* - *message severity* - *message text*

Example 23–1 shows an extract from an error log.

EXAMPLE 23–1 Extract of an Error Log

```
[07/26/2005:10:41:38 +0200] - STARTUP    - INFO  - Sun Java(TM)
  System Directory Proxy Server/6.1 (Build 0719051656 DEBUG) starting up
[07/26/2005:10:41:43 +0200] - STARTUP    - INFO  - Global log level INFO
[07/26/2005:10:41:43 +0200] - STARTUP    - INFO  - Logging Service conf
[07/26/2005:10:41:43 +0200] - STARTUP    - INFO  - Java Version: 1.5.0_03
 (Java Home: /usr/lang/JAVA/jdk1.5.0_03/solaris-sparc/jre)
[07/26/2005:10:41:43 +0200] - STARTUP    - INFO  - Operating System:
[07/26/2005:10:41:43 +0200] - STARTUP    - INFO  - Init LDAP server
  cn=Server-1, cn=LDAP servers,cn=config
[07/26/2005:10:41:43 +0200] - STARTUP    - INFO  - Init LDAP server
  cn=Server-2,cn=LDAP servers,cn=config
[07/26/2005:10:41:43 +0200] - STARTUP    - INFO  - Init LDAP server
  cn=Server-3, cn=LDAP servers,cn=config
[07/26/2005:10:41:44 +0200] - STARTUP    - INFO  - Performing SSL init
[07/26/2005:10:41:44 +0200] - STARTUP    - INFO  - Creating 20 worker
  threads
[07/26/2005:10:41:44 +0200] - PLUGIN     - WARN  - Unable to load
  plugin class
  com.sun.directory.proxy.plugin.StartTLSExtendedOpPlugin specified in
  plugin
  entry cn=1.3.6.1.4.1.1466.20037,cn=Plugins,cn=config -- not loading
  plugin.
[07/26/2005:10:41:45 +0200] - STARTUP    - INFO  - Starting client
  listeners
[07/26/2005:10:41:45 +0200] - STARTUP    - INFO  - Sun Java(TM)
  System Directory Proxy Server/6.1 (Build 0719051656 DEBUG) started
  on host a
[07/26/2005:10:41:45 +0200] - STARTUP    - INFO  - Listening for secure
  client connections on 0.0.0.0:9636
[07/26/2005:10:41:45 +0200] - STARTUP    - INFO  - Listening for client
  connections on 0.0.0.0:9389
[07/26/2005:10:42:13 +0200] - BACKEND    - WARN  - Proactive Monitor
  thread determined that directory server ldap://nautilus:6389/ avail
[07/26/2005:10:42:13 +0200] - BACKEND    - WARN  - Proactive Monitor
  thread determined directory server ldap://nautilus:7389/ is available.
[07/26/2005:10:42:13 +0200] - BACKEND    - WARN  - Proactive Monitor
  thread determined directory server ldap://nautilus:5389/ is available.
```

Access Logs for Directory Proxy Server

Access logs contain information about the requests being processed by Directory Proxy Server. Access logs contain information about two types of connection:

- Connections between clients and Directory Proxy Server
- Connections between Directory Proxy Server and data sources

Access log messages are categorized according to the cause of the message. The following table lists the categories of messages that can be included in the access log.

TABLE 23–2 Message Categories for Access Logs

Category Name	Category Description
`CONNECT`	Information about a client connection
`DISCONNECT`	Information about a client disconnection
`OPERATION`	Information about operations requested by a client
`PROFILE`	Information about the profiles of a connection handler
`SERVER_OP`	Information about operations that are forwarded to data sources
`SERVER_OP_DETAIL`	Detailed information about operations that are forwarded to data sources

Access Log Levels

Each message category can be configured with one of the following log-levels:

1. `none` No access messages are included in the log file.
2. `info` Informational messages are included in the log file.
3. `all` All messages are included in the log file. In most cases, this setting produces the same results as the `info` setting. In certain situations, this setting enables additional debugging messages to be logged.
4. `inherited` The log level is inherited from the value of the `default-log-level` property.

By default, the log level for each message category is `info`.

The log-level of a message category works in conjunction with the severity level of a message to determine which messages are included in the log file. For more information, see "Message Severity" on page 345.

Format of an Access Log Message

An access log message has this format:

timestamp - *category* - *severity* - *connectionNumber operationNumber*
  messageID operationType messageText

Example 23–2 shows an extract of an access log. The log shows a client request that starts with a message in the CONNECT category and ends with a message in the DISCONNECT category. The operation requested by the client is shown by the message in the OPERATION category, and results in several messages in the SERVER_OP category. The logged messages have the INFO and DEBUG severity.

EXAMPLE 23–2 Extract of an Access Log

```
[07/17/2005:17:29:45 +0200] - CONNECT    - INFO  - conn=1591031
  client=129.157.192.132:49216 server=0.0.0.0:9389 protocol=LDAP
[07/17/2005:17:29:45 +0200] - OPERATION  - INFO  - conn=1591031 op=0
  msgid=1 SEARCH base="o=movie" scope=2 filter="(objectClass=*)"
[07/17/2005:17:29:45 +0200] - SERVER_OP  - INFO  - conn=1591031 op=0
  SEARCH base="o=movie" scope=2 filter="(objectClass=*)"
  s_msgid=318022 s_authzid="" s_conn=39
[07/17/2005:17:29:45 +0200] - SERVER_OP  - INFO  - conn=1591031 op=0
  SEARCH base="o=movie" scope=2 filter="(objectClass=*)" s_msgid=316902
  s_authzid="" s_conn=76
[07/17/2005:17:29:45 +0200] - SERVER_OP  - INFO  - conn=1591031 op=0
  SEARCH RESPONSE err=0 msg="" nentries=4 s_conn=76
[07/17/2005:17:29:45 +0200] - SERVER_OP  - DEBUG - Global status code = 0
[07/17/2005:17:29:45 +0200] - SERVER_OP  - INFO  - conn=1591031 op=0
  SEARCH RESPONSE err=0 msg="" nentries=11 s_conn=39
[07/17/2005:17:29:45 +0200] - SERVER_OP  - DEBUG - Global status code = 0
[07/17/2005:17:29:45 +0200] - OPERATION  - INFO  - conn=1591031 op=0
  SEARCH RESPONSE err=0 msg="" nentries=22
[07/17/2005:17:29:45 +0200] - OPERATION  - INFO  - conn=1591031 op=1
  UNBIND
[07/17/2005:17:29:45 +0200] - DISCONNECT - INFO  - conn=1591031
  reason=unbind"
```

Message Parts in an Access Log

Messages for the connections between a client and the Directory Proxy Server are labeled in the same way as in Directory Server. Table 23–4 describes parts of the messages between the client and the Directory Proxy Server in Example 23–2. For an explanation of all of the possible message parts, see "Content of Access, Error, and Audit Logs" on page 164.

TABLE 23–3 Message Parts for Connections Between a Client and a Directory Proxy Server

Log Message Part	Description
conn	Identifier for the connection between the client and the Directory Proxy Server.
op	The number of an operation on a given connection. The first operation on a connection has the value op=0. Subsequent requests on the connection have increasing numbers, op=1, op=2, etc.
msgid	The number of a message to be sent to a client application. The LDAP protocol is mainly asynchronous. If a client request requires a response from a server, the response is given in the following steps: ■ The directory server acknowledges the request and assigns a msgid ■ The directory server responds to the request by using the msgid identifier A response can be sent in multiple packets, where each packet is identified by the same msgid.
nentries	The number of entries returned by a search request.
err	The result code returned from the LDAP operation. The error number 0 means that the operation was successful. For a list of LDAP result codes, see "Result Codes in Log Files" on page 170.
msg	A human readable error diagnostic.

Messages for the connections between Directory Proxy Server and a data source are prefixed with s_. Table 23–4 describes parts of the messages between the Directory Proxy Server and the data source in Example 23–2.

TABLE 23–4 Message Parts for Connections Between a Directory Proxy Server and a Data Source

Log Message Part	Description
s_msgid	Identifier for the message between the Directory Proxy Server and a data source.
s_authzid	Authorization identity for an operation to be processed under when the Directory Proxy Server forwards the request to a data source by using proxy authorization.
s_conn	Identifier for the connection between the Directory Proxy Server and the data source.

Access Log Buffer

Access log messages are stored in a buffer. The buffer is flushed to the access log at the following times:

- When the buffer is full
- When the access log is rotated
- When Directory Proxy Server is stopped

If a buffer is flushed because it is full, the last message in the access log file might not be complete. The remainder of the message is then delivered in the next flush. By default, the size of the buffer is 10 KBytes. However, the size of the buffer can be configured to control the frequency with which it is flushed. For performance reasons, the buffer size should not be reduced to less than 5 KBytes.

You can configure the size of the access log buffer by setting the log-buffer-size property. For information about how to configure access log properties, see "Configuring Directory Proxy Server Logs" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Tracking Client Requests Through Directory Proxy Server and Directory Server Access Logs

Access logs show client accesses to the server and corresponding server responses. Directory Proxy Server access logs further show information about the connections set up against data sources, in this case Directory Server instances.

Tracking client requests can be broken down into the following steps:

- Tracking the operations performed within a single client connection
- Identifying the client that performed a certain operation

Tracking Operations by Connection

Directory Proxy Server typically sets up connections with backend servers before it handles client connections. This means that the proxy can pool operations, binding and rebinding only when necessary and avoiding connection setup overhead. Directory Proxy Server identifies these backend connections in its access log with tags of the form `s_conn=data-source:number`, where `data-source` is a data source name from the configuration and `number` is a server connection number assigned by the proxy. Such `s_conn` server connections can then be matched to connection numbers in Directory Server access logs using the port number from which the proxy connected to the directory as a client when establishing the connection. Therefore, `s_conn` in proxy access log messages be translated into `conn` in directory access log messages.

Tracking Operations in Directory Proxy Server

In the Directory Proxy Server access log, each client operation is contained within a `CONNECT` and a `DISCONNECT` message. Between these two messages, several `OPERATION` messages can appear. Each `OPERATION` message can contain several `SERVER_OP` messages.

The `OPERATION` messages refer to operations performed by the client. The `SERVER_OP` messages refer to operations performed by Directory Proxy Server.

The following extract of a Directory Proxy Server access log file shows the start (CONNECT) and end (DISCONNECT) of a connection, conn=0. The log shows all the OPERATION requests performed by a client this connection and the related SERVER_OP requests sent to the backend server by Directory Proxy Server on behalf of the client.

```
[timestamp] - CONNECT    - INFO  - conn=0 client=129.157.192.132:59112 server=0.0.0.0:9389 protocol=LDAP
[timestamp] - OPERATION  - INFO  - conn=0 op=0 BIND dn="uid=u1,ou=users,o=movie" method="SIMPLE"
[timestamp] - SERVER_OP  - INFO  - conn=0 op=0 BIND dn="uid=u1,ou=users,o=movie" method="SIMPLE" s_msgid=2
                                                    s_conn=server-1:1
[timestamp] - SERVER_OP  - INFO  - conn=0 op=0 BIND RESPONSE err=0 msg="" s_conn=server-1:1
[timestamp] - OPERATION  - INFO  - conn=0 op=0 BIND RESPONSE err=0 msg="" etime=0
[timestamp] - OPERATION  - INFO  - conn=0 op=1 msgid=2 SEARCH base="o=movie" scope=2 filter="(objectclass=*)"
[timestamp] - SERVER_OP  - INFO  - conn=0 op=1 SEARCH base="o=movie" scope=2 filter="(objectclass=*)" s_msgid=3
                                                      s_conn=server-1:1
[timestamp] - SERVER_OP  - INFO  - conn=0 op=1 SEARCH RESPONSE err=0 msg="" nentries=12 s_conn=server-1:1
[timestamp] - OPERATION  - INFO  - conn=0 op=1 SEARCH RESPONSE err=0 msg="" nentries=12 etime=0
[timestamp] - OPERATION  - INFO  - conn=0 op=2 UNBIND
[timestamp] - SERVER_OP  - INFO  - conn=0 op=-1 BIND dn="" method="SIMPLE" s_msgid=4 s_conn=server-1:1
[timestamp] - SERVER_OP  - INFO  - conn=0 op=-1 BIND RESPONSE err=0 msg="" s_conn=server-1:1
[timestamp] - DISCONNECT - INFO  - conn=0 reason="unbind"
```

Following this log, it is possible to track all operations that were performed by or on behalf of a particular client.

Tracking Operations Between Directory Proxy Server and Directory Server

When Directory Proxy Server starts up, it establishes connections with all the remote servers identified in its configuration. These connections are logged in the Directory Proxy Server access log, and are identified by the field s_conn=*server-name*:*number*. The *server-name* is defined in the Directory Proxy Server configuration and refers to a specific backend server. The *number* indicates how many connections there have been to this backend server, through the same port.

For example, in the following extract from the Directory Proxy Servers_conn=server-1:1 is the first connection to remote server server-1 through port 59100.

```
SERVER_OP - INFO - Created connection for BIND s_conn=server-1:1 client=129.157.192.132:59100
```

When this connection is established, the corresponding line in the Directory Server access log shows that the connection from Directory Proxy Server through port 59100 is identified with the connection ID conn=244.

```
conn=244 op=-1 msgId=-1 - fd=19 slot=19 LDAP connection from 129.157.192.132:59100 to 129.157.192.132
```

For the remainder of the life of this connection, server-1:1 in the Directory Proxy Server can be mapped to conn=24 in the Directory Server access log.

This kind of mapping between connections also requires that Directory Proxy Server and the backend Directory Server are synchronized.

Note that a connection from Directory Proxy Server to a backend Directory Server can remain alive for several days. If you rotate logs, either manually or automatically, it might therefore be necessary to access archived log files to trace the operations performed during a connection.

Client Identification

A client is identified in the access logs by its IP address and, optionally, by its bind DN. When a client establishes a connection to Directory Proxy Server, the following kind of message is logged in the Directory Proxy Server access log:

```
[timestamp] - CONNECT  - INFO  - conn=0 client=IP1:port1 server=IP2:port2 protocol=LDAP
```

Directory Proxy Server identifies this client connection as **conn=0**.

When Directory Proxy Server establishes a connection with a remote Directory Server, the following kind of message is logged in the Directory Proxy Server access log:

```
[timestamp] - SERVER_OP  - INFO  - Created connection for READ s_conn=server-1:1 client=IP2:port3
 server=IP4:port4 protocol=LDAP main
```

Directory Proxy Server identifies this connection to the remote server as **s_conn=server-1:1**.

At the same time, the following kind of message is logged in the Directory Server access log:

```
[timestamp] conn=13 op=-1 msgId=-1 - fd=23 slot=23 LDAP connection from IP2:port3 to IP4
```

So, Directory Server identifies the connection as **conn=13**.

Tracking the connection in this way enables you to identify the full connection path from the client to Directory Server.

Directory Proxy Server does not wait for a client connection before it establishes a connection to a remote server. The Directory Proxy Server configuration specifies that certain connections are dedicated to bind operations, others to read operations, and others to write operations. When Directory Proxy Server starts up, it establishes all connections to the remote servers, according to this configuration.

When a connection has been established completely (from the client to Directory Server) the client can be identified by its DN.

Directory Server recognizes the client DN as one of the following:

- **True client bind DN.** The bind DN is the client's own bind DN if Directory Proxy Server is configured in Use Bind mode.
- **Modified client bind DN.** The bind DN is modified if Directory Proxy Server is configured in User Proxy Auth Control mode. The DN is modified as a result of DN renaming or user mapping.

A single connection can be used by multiple clients (though not simultaneously). To identify a client connection correctly in the access logs, Directory Proxy Server and Directory Server must be synchronized, that is, the server clock must be as close as possible. This will ensure that the timestamps in the access logs correspond. If the servers are not synchronized, you should synchronize them by using a time server, or evaluate the difference between the server clocks and search the access logs taking this difference into account.

CHAPTER 24

Directory Proxy Server Alerts and Monitoring

The Directory Proxy Server provides monitoring information about its own status. Directory Proxy Server also monitors data sources to determine whether they are alive and to detect failed connections. If a data source fails, Directory Proxy Server can switch new requests over to a working data source in a data source pool and can replay failed requests to this new data source.

This chapter describes how monitoring is implemented in Directory Proxy Server. The chapter covers the following topics:

- "Administrative Alerts for Directory Proxy Server" on page 355
- "Monitoring Data Sources" on page 356
- "Monitoring Directory Proxy Server" on page 359

Administrative Alerts for Directory Proxy Server

Directory Proxy Server generates a set of predefined administrative alerts. You can select one or more of the predefined administrative alerts and configure Directory Proxy Server to take a specific action when the alert events occur:

The actions that can be taken include the following:

- Create a syslog entry. Alerts are sent to the syslog with the facility of USER.
- Send an e-mail message.
- Run a script command.

Table 24–1 lists the predefined administrative alerts for Directory Proxy Server.

TABLE 24–1 Administrative Alerts for Directory Proxy Server

Alert event	Alert code	Configuration Parameter
Server startup	`1000`	`info-server-startup`

TABLE 24–1 Administrative Alerts for Directory Proxy Server *(Continued)*

Alert event	Alert code	Configuration Parameter
Clean server shutdown	`1001`	`info-server-shutdown-clean`
Abrupt server shutdown	`1002`	`error-server-shutdown-abrupt`
Configuration reloaded	`1003`	`info-configuration-reload`
Configuration reload failure due to bad configuration. Run-time configuration not impacted.	`1004`	`warning-configuration-reload-failure-no-impact`
Configuration reload failure due to bad configuration. Run-time configuration possibly impacted.	`1005`	`error-configuration-reload-failure-with-impact`
Data source not available	`2000`	`warning-data-source-unavailable`
Data source available	`2001`	`info-data-source-available`
Listener not available	`3000`	`warning-listener-unavailable`
Data inconsistency on data sources	`4000`	`warning-data-sources-inconsistent`

For information about how to configure administrative alerts for Directory Proxy Server, see "Configuring Administrative Alerts for Directory Proxy Server" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Monitoring Data Sources

Directory Proxy Server continuously monitors data sources to determine whether they are alive and to detect failed connections. This section describes how Directory Proxy Server monitors data sources, and what action is taken when data sources fail.

How Data Sources Are Monitored

Directory Proxy Server performs the following tests to monitor the health of a data source:

- Listens for errors on the traffic between Directory Proxy Server and the data source
- Periodically establishes a dedicated connection to the data source if there is no traffic from that data source for a specified time interval
- Periodically pings each existing connection to each data source to prevent that connection from being closed and to detect closed connections

These tests are described in the following sections.

Monitoring a Data Source by Listening for Errors

When this type of monitoring is configured, Directory Proxy Server listens for errors on the traffic between itself and the data source. If Directory Proxy Server detects that a client operation fails, the proxy tests the data source related to the failure.

This type of monitoring is called *reactive monitoring* because Directory Proxy Server reacts to an error, but otherwise performs no active testing of the data sources.

Directory Proxy Server can be configured to perform this type of reactive monitoring only, without performing the monitoring described in "Monitoring Data Sources by Periodically Establishing Dedicated Connections" on page 357 and "Monitoring Data Sources by Testing Established Connections" on page 357. When only reactive monitoring is configured, the monitoring less complete but does not cause additional traffic.

Monitoring Data Sources by Periodically Establishing Dedicated Connections

When this type of monitoring is configured, Directory Proxy Server establishes a dedicated connection to a data source when no requests made to the data source or responses given by the data source for a specified time period. By periodically establishing a dedicated connection to a data source, Directory Proxy Server monitors whether the data source is working.

This type of monitoring is more complete than "Monitoring a Data Source by Listening for Errors" on page 357 because Directory Proxy Server does not wait to detect a failure before it tests the data source. However, this type of monitoring is less complete than "Monitoring Data Sources by Testing Established Connections" on page 357, because the proxy does not test whether the existing connections to a data source are working.

This type of monitoring can be used in addition to "Monitoring Data Sources by Testing Established Connections" on page 357.

Monitoring Data Sources by Testing Established Connections

When this type of monitoring is configured, Directory Proxy Server tests each connection to each data source at regular intervals. In this way, the proxy prevents connections from being dropped because of inactivity, and detects closed connections.

This type of monitoring can be used in addition to "Monitoring Data Sources by Periodically Establishing Dedicated Connections" on page 357.

Directory Proxy Server can be configured to test connections in the following scenarios:

- Pooled connections that are not used for a period of time
- Connections for persistent searches that are not active for a period of time
- Connections between a client and Directory Proxy Server operating in tunneling mode

Testing established connections consumes system resources, but it provides good security for connections. If you are using the Active Directory product, you must use this method of monitoring because the Active Directory product closes inactive connections.

To test an established connection, Directory Proxy Server issues a search request with the following parameters:

- Search base DN
- Connection time out
- Search time out
- Search filter

If a connection is found to be down, Directory Proxy Server polls the connection at a specified interval to detect its recovery. You can configure this interval by setting the `monitoring-interval` property. For more information, see "To Monitor a Data Source by Testing Established Connections" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Directory Proxy Server monitors data sources by using a search filter. Data sources that return a result that satisfies the filter are considered to be working.

Responding to the Failure of a Data Source

When Directory Proxy Server detects an error on a connection, the proxy closes the connection and tries to reestablish the connection immediately. If the proxy can reestablish the connection, it considers the data source to be up and running. If the proxy cannot reestablish the connection, it flags the data source as unavailable. Directory Proxy Server stops distributing requests to the data source and closes all other connections to the data source.

If a request fails because of a failed connection or a failed data source, Directory Proxy Server replays the request over another connection to the same data source or replays the request to another data source. If the request is replayed to another data source, the load balancing algorithm determines which data source is used.

If there are no data sources to which Directory Proxy Server can replay the request, the proxy returns an error to the client.

Replaying the request enables the failure to be transparent to the client. Requests are replayed for the following operations:

- Search
- Bind
- Compare

Requests are not replayed for write operations because Directory Proxy Server cannot be sure whether the operation was performed before the connection failure occurred.

When a data source recovers after a being unavailable, Directory Proxy Server returns the data source to the list of candidate data sources. The work that was being carried out by the other candidate data sources is redistributed to include this data source, according to the load balancing algorithm.

When the failed data source recovers, Directory Proxy Server recommences monitoring the traffic between the data sources and their clients.

Monitoring Directory Proxy Server

Directory Proxy Server runs inside a Java Virtual Machine (JVM) and depends on the memory of the JVM. To ensure that Directory Proxy Server is running correctly, its memory consumption must be monitored. For information about how to monitor Directory Proxy Server memory consumption, see "Retrieving Monitored Data About Directory Proxy Server by Using the JVM" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Monitoring information for Directory Proxy Server is provided under the `cn=monitor` entry. The `cn=monitor` entry is managed by Directory Proxy Server in a local, in-memory database.

For information about monitoring Directory Proxy Server, see the following sections:

- "Monitoring Framework for Directory Proxy Server" on page 359
- "Simplified Layout of the `cn=monitor` Entry" on page 360
- "Status of Monitored Information" on page 362
- "Description of Each Entry Under the `cn=monitor` Entry" on page 362
- "Detailed Layout of the `cn=monitor` Entry" on page 373

Monitoring Framework for Directory Proxy Server

Directory Proxy Server monitoring relies on the Java Enterprise System (ES) Monitoring Framework. The Java ES monitoring framework has been extended to provide a monitoring framework for Directory Proxy Server. The following UML diagram illustrates the Directory Proxy Server monitoring framework.

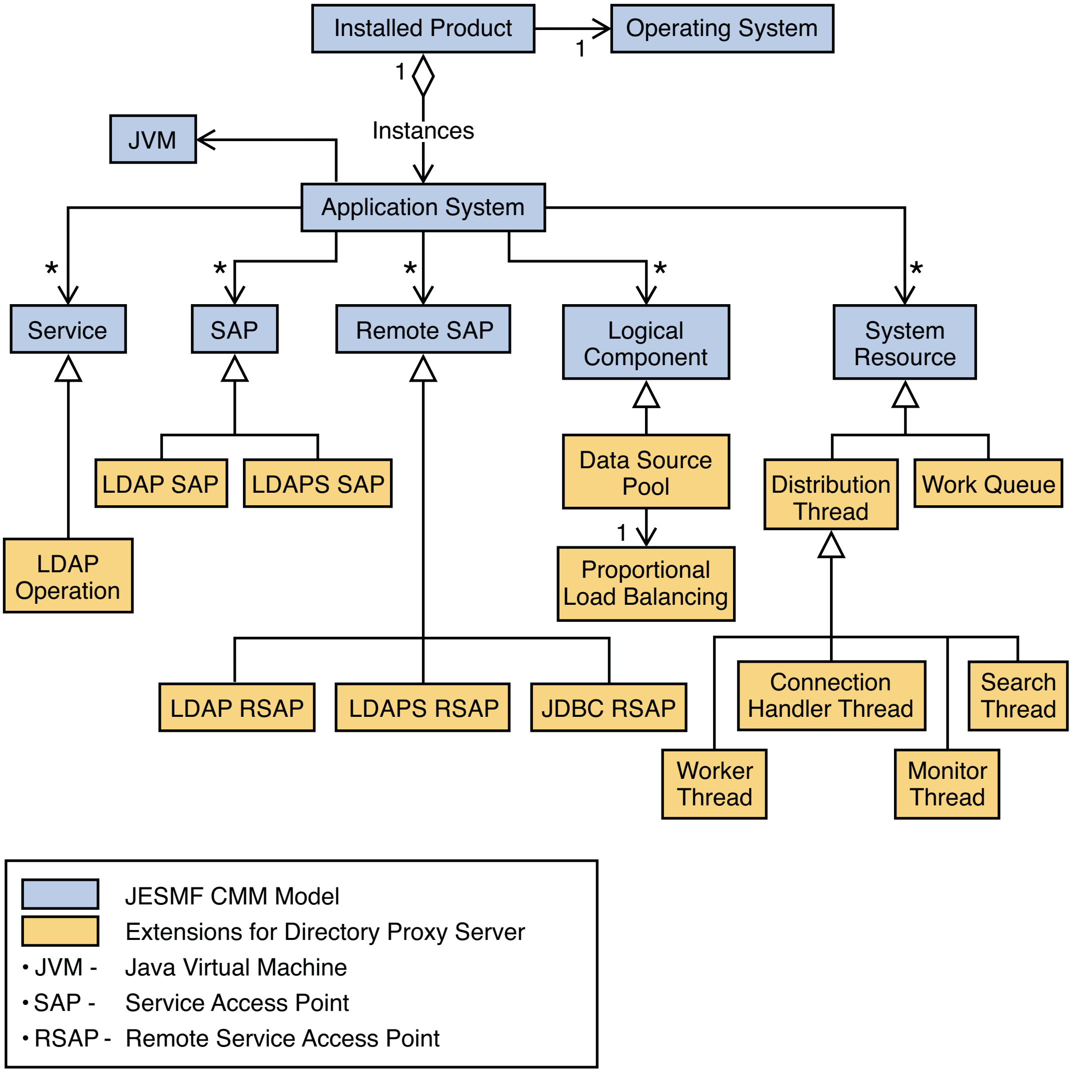


FIGURE 24–1 Monitoring Framework for Directory Proxy Server

Simplified Layout of the `cn=monitor` Entry

This section provides a simplified layout of the `cn=monitor` entry. For the detailed layout of the `cn=monitor` entry and a description of all of the entries and attributes under `cn=monitor`, see "Detailed Layout of the `cn=monitor` Entry" on page 373.

```
cn=monitor
|
+-- cn=Product (Installed Product)
    |
    +-- cn=ProductName
        |
        +-- cn=Operating System
        +-- cn=Instance (Application System)
            |
            +-- cn=InstanceId
                |
                +-- cn=Service
                    +-- cn=Add
                    +-- cn=Delete
                    +-- cn=Modify
                    +-- cn=ModifyDN
                    +-- cn=Search
                    +-- cn=Compare
                    +-- cn=Bind
                +-- cn=Resource (System Resource)
                    +-- cn=Work Queue
                    +-- cn=Worker Thread
                        +-- cn=worker_thread_name
                    +-- cn=Search Thread
                        +-- cn=search_thread_name
                    +-- cn=Monitor Thread
                        +-- cn=monitor_thread_name
                    +-- cn=Connection Handler Thread
                        +-- cn=connection_handler_thread_name
                +-- cn=SAP (Service Access Point)
                    +-- cn=LDAP
                    +-- cn=LDAPS
                +-- cn=RSAP (Remote SAP)
                    +-- cn=LDAP  Server servername
                    +-- cn=LDAPS Server servername
                    +-- cn=RDBM  Server servername
                +-- cn=Component (Logical Component)
                    +-- cn=DataSource Pool
                        +-- cn=poolname
                            +-- cn=Proportional Load Balancing
                                +-- cn=Add
                                +-- cn=Search
                                +-- cn=Delete
                                +-- cn=Compare
                                +-- cn=Modify
                                +-- cn=ModifyDN
                                +-- cn=Bind
                +-- cn=JVM
                +-- cn=DB System
                +-- cn=DB Service
```

Status of Monitored Information

Every element that is monitored has an operational status. Table 24–2 gives the status of monitored information.

TABLE 24–2 Status of Monitored Information

Value	Name	Description
0	`UNKNOWN`	No information available for this element
2	`OK`	Element is fully operational
3	`DEGRADED`	Element is working but not optimally
4	`STRESSED`	Element is working under stressed environment (for example, overload)
5	`PREDICTIVE_FAILURE`	Element is working but may fail in the near future
6	`ERROR`	Severe error has been raised
8	`STARTING`	Element is starting
9	`STOPPING`	Element is stopping
10	`STOPPED`	Element is stopped
12	`NO_CONTACT`	Element has never been reached
13	`LOST_COMMUNICATION`	Element has been reached once, but it is currently unreachable

Description of Each Entry Under the `cn=monitor` Entry

For information about each entry in the `cn=monitor` subtree, see the following sections:

- "`cn=Product`" on page 363
- "`cn=Operating System`" on page 363
- "`cn=Instance`" on page 363
- "`cn=Service`" on page 364
- "`cn=SAP`" on page 364
- "`cn=RSAP`" on page 365
- "`cn=Component`" on page 367
- "`cn=JVM`" on page 369
- "`cn=Resource`" on page 369

cn=Product

The cn=Product entry identifies the set of files being installed. An installed product is identified by the entry cn=ProductName.

cn=Product contains the following groups of attributes:

Settings

- version identifies the full release number containing major release, minor release and optionally micro release (for example, 6.1).
- buildNumber identifies the syntax of the build number.
- patchId identifies the patch of the product. This attribute can be empty.

State Provides operational status and availability status.

Statistics Provides a set of statistics metrics such as performance and usage.

cn=Operating System

The cn=Operating System entry identifies which operating system the product package is installed under. This entry has the following monitoring DN:

```
cn=Operating System, cn=ProductName, cn=Product, cn=monitor
```

cn=Operating System contains the following groups of attributes:

Settings

- operatingSystemName identifies the name of the operating system such as SunOS.
- operatingSystemVersion identifies the release of the operating system such as 5.10.

State Provides operational status and availability status.

Statistics Provides a set of statistics metrics such as performance and usage.

cn=Instance

The cn=Instance entry identifies an instance of the installed product. More than one instance of a product can exist on a single data source. Each instance is identified by an instance ID, where *instanceId=host:instance-path*.

The cn=Instance entry has the following monitoring DN:

```
cn=InstanceId, cn=Instance, cn=Operating System, cn=ProductName,
  cn=Product, cn=monitor
```

cn=Operating System contains the following groups of attributes:

Settings Provides configuration attribute values.

State operationalStatus identifies the status of the element, with the following values: 0, 2, 8, 9, and 10. For information about the values, see Table 24–2.

Statistics Provides a set of statistics metrics such as performance and usage.

cn=Service

The cn=Service entry identifies information about LDAP operations, or services, for an instance of Directory Proxy Server.

LDAP operations are add, delete, modify, modifyDN, search, compare, and bind. Each LDAP operation has a specific monitoring entry below cn=Service. For example, the add operation has the following DN:

```
cn=add, cn=Service, cn=InstanceId, cn=Instance, cn=Operating System,
  cn=ProductName, cn=Product, cn=monitor
```

Entries below cn=Service can contain the following groups of attributes:

Settings Provides configuration attribute values.

State Provides operational status and availability status.

Statistics

- total identifies the number of operations received by this instance of Directory Proxy Server.
- succeeded identifies the number of successful operations in this instance of Directory Proxy Server.
- failed identifies the number of failed operations in this instance of Directory Proxy Server.
- abandonned identifies the number of operations abandoned by this instance of Directory Proxy Server.

cn=SAP

A Service Access Point (SAP) provides information on how to access a service. The cn=SAP entry has the following monitoring DN:

```
cn=listenerThread, cn=SAP, cn=instanceId, cn=Instance,
  cn=OperatingSystem, cn=Product, cn=monitor
```

Entries below cn=SAP can contain the following groups of attributes:

Settings

- name identifies the SAP name, either LDAP or LDAPS.
- isSecure identifies whether LDAPS is used. If the value is TRUE, LDAPS is used.
- host identifies the hostname of the current data source.
- port identifies the port number to access this instance of Directory Proxy Server.

State

- enabled identifies if the SAP is enabled.
- operationalStatus identifies the status of the SAP. If the value is 2 or OK, the SAP is fully operational.
- statusDescription provides a detailed status description.
- startTime identifies the date and time at which the SAP was started.
- stopTime identifies the date and time at which the SAP was stopped.
- stopException provides a description of the error if a stop operation fails. If this attribute is empty, no error has occurred.

Statistics

- acceptedConnections identifies the number of accepted TCP connections. One counter exists for all LDAP operations. The counter is service agnostic.
- refusedConnections identifies the number of refused TCP connections.

cn=RSAP

The cn=RSAP entry identifies the type of remote service. The remote SAP can be one of the following types:

- LDAP(S) to access directory servers such as Sun Directory Server or Microsoft ADAM.
- ODBC to access RDBM systems such as the Oracle Database.

LDAP Remote SAP

The cn=RSAP entry for an LDAP remote SAP can have one of the following monitoring DNs:

```
cn=LDAP  servername, cn=RSAP, cn=instanceId, cn=Instance,
  cn=OperatingSystem, cn=Product, cn=monitor

cn=LDAPS  servername, cn=RSAP, cn=instanceId, cn=Instance,
  cn=OperatingSystem, cn=Product, cn=monitor
```

Entries below cn=RSAP can contain the following groups of attributes:

Settings

- `name` identifies the SAP name, either `LDAP` or `LDAPS`.
- `isSecure` identifies whether `LDAPS` is used. If the value is `TRUE`, `LDAPS` is used.
- `host` identifies the hostname of the host server.
- `port` identifies the port number to access this instance of Directory Proxy Server.

State

- `operationalStatus` identifies the status of the element, with the following values: `2`, `4`, `12`, and `13`. For information about these values, see Table 24–2.
- `statusDescription` provides the detailed description of the status.
- `started` identifies if RSAP is started.
- `readOnly` identifies if it is in read only mode.

Statistics

- `totalConnections` identifies the total number of connections including the established connections.
- `totalAvailableConnections` identifies the total number of available connections for bind, read and write operations. The value `0` means that access to that data source is congested.
- The following attributes are given for bind operations but also exist for read operations and write operations:
  - `totalBindConnections` identifies the number of established connections for bind operations. All of the connections are kept in a pool of bind connections.
  - `availableBindConnections` identifies the number of free bind connections in the pool.
  - `bindConnectionsRequested` identifies the number of requests to get a free bind connection from the pool.
  - `bindConnectionsProvided` identifies the number of bind connections provided upon request.
  - `bindConnectionsRefused` identifies the number of requests being refused because the pool is empty (even after a wait) or because the remote data source is down.
  - `bindConnectionsWaitsRequired` identifies the number of requests being blocked in a wait state, waiting for a bind connection to be freed.
  - `bindConnectionsReturnedValid` identifies the number of connections being released.

- `bindConnectionsReturnedInvalid` identifies the number of connections being released as invalid. A connection is said to be invalid when errors have occurred.

`cn=Component`

The `cn=Component` entry identifies the part of the software being accessed through a service. The following parts of the software are identified by the `cn=Component` entry:

- Load balancing algorithm
- Connection class
- Data view

Proportional Load Balancing Algorithm For All Data Sources

The `cn=Component` entry for the proportional load balancing algorithm for all data sources has the following monitoring DN:

```
cn=ProportionalLB, cn=DataSourcePool poolname, cn=Component,
  cn=instanceId, cn=Instance, cn=OperatingSystem, cn=Product, cn=monitor
```

Entries below the `cn=Component` entry for the proportional load balancing algorithm contain the following groups of attributes for all data sources:

Settings

- `className` provides the name of the class.

State

- `enabled` identifies the status of the remote SAP. If the value is `TRUE`, the load balancing algorithm is active.

Statistics

- `totalBindConnectionsProvided` identifies the total number of connections provided for bind operations.
- `totalBindConnectionsRefused` identifies the number of refused connections for bind operations.

  Connections can be refused for one of the following reasons:

  - The pool of data sources is empty.
  - All the data sources in the pool are down.
  - The data source selected by the load balancing algorithm has no free connection to reach the remote service.
- `totalAddConnectionsProvided` see `totalBindConnectionsProvided`

- totalAddConnectionsRefused see totalBindConnectionsRefused
- totalCompareConnectionsProvided see totalBindConnectionsProvided
- totalCompareConnectionsRefused see totalBindConnectionsRefused
- totalDeleteConnectionsProvided see totalBindConnectionsProvided
- totalDeleteConnectionsRefused see totalBindConnectionsRefused
- totalModifyConnectionsProvided see totalBindConnectionsProvided
- totalModifyConnectionsRefused see totalBindConnectionsRefused
- totalModifyDNConnectionsProvided see totalBindConnectionsProvided
- totalModifyDNConnectionsRefused see totalBindConnectionsRefused
- totalCompareConnectionsProvided see totalBindConnectionsProvided
- totalCompareConnectionsRefused see totalBindConnectionsRefused

Proportional Load Balancing Algorithm For Individual Data Sources

The cn=Component entry for the proportional load balancing algorithm for individual data sources has the following monitoring DN:

```
cn=Add, cn=servername, cn=Proportional LB, cn=DataSource Pool poolname,
  cn=Component, cn=instanceId, cn=Instance, cn=OperatingSystem,
  cn=Product, cn=monitor
```

Similar monitoring DNs exist for the delete, modify, modifyDN, search, compare, and bind operations.

Entries below the cn=Component entry for the proportional load balancing algorithm contain the following groups of attributes for individual data sources:

Settings

- Provides configuration attribute values.

State

- operationalStatus identifies the status of the element, with the following values: 2, and 5. For information about these values, see Table 24–2.
- statusDescription provides the detailed status description.

Statistics

- providedConnections the number of connections provided to reach the data source for the operation.
- providedPercentage the percentage of connections provided to reach the data source for the operation.

- refusedConnections the number of refused requests to get a connection to that data source.
- refusedPercentage the percentage of refused requests.

cn=JVM

The cn=JVM entry identifies the JVM that is being used to run the instance of Directory Proxy Server. The cn=JVM entry has the following monitoring DN:

```
cn=JVM, cn=instanceId, cn=Instance, cn=DPS60, cn=Product, cn=monitor
```

Entries below cn=JVM can contain the following groups of attributes:

Settings

- version identifies the version of the JVM used to run the instance of Directory Proxy Server.
- JVMInstallation identifies the location of the JVM installation.

State

- operationalStatus identifies the status of the element, with the following values: 2, and 5. For information about these values, see Table 24–2.
- statusDescription provides the detailed status description.

Statistics

- totalJVMMemory identifies the total amount of memory allocated for the JVM to run.
- maxJVMMemory identifies the maximum amount of JVM memory.
- freeJVMMemory identifies the amount of free memory.
- realFreeJVMMemory identifies the free JVM memory which can be used.
- JVMMemoryLowLevelCount provides the number of times JVM memory changes its state from green to orange.
- JVMMemoryVeryLowLevelCount provides the number of times JVM memory changes its state from orange to red.
- availableCPU identifies the CPU capacity available.

cn=Resource

The cn=Resource entry identifies the set of resources being used by the software. Resources include buffers, file descriptors, and hard disks.

The following elements are identified by the cn=Resource entry:

- “Connection Handler Thread” on page 370
- “Work Queue” on page 371
- “Worker Thread” on page 372
- “Search Thread” on page 372
- “Monitor Thread” on page 373

Connection Handler Thread

The connection handler thread decodes incoming requests. The connection handler is oriented to the LDAP or LDAPS protocol. When a request has been fully decoded, the request is put in the work queue.

The cn=Resource entry for the connection handler thread has the following monitoring DN:

```
cn=connection_handler_thread_name, cn=Connection Handler Thread,
  cn=Resource, cn=instanceId, cn=Instance, cn=DPS60, cn=Product,
  cn=monitor
```

Entries below the cn=Resource entry for the connection handler thread contain the following groups of attributes:

Settings

- threadID provides the unique thread identification number.
- threadStack provides the information on threads stack.

State

- operationalStatus identifies the status of the element. The value 2 indicates that the element is fully operational.
- startTime identifies the date and time at which the thread was started.
- started identifies if the thread has started.
- running identifies if the thread is in running state.
- statusDescription provides the detailed status description.

Statistics

The following statistics can be gathered:

- Byte buffer pool statistics under cn=ByteBufferPool:
  - numTries
  - numHits
  - numMissesEmpty
  - numMissesSize
  - numReleases

  - availableStandardBuffers
  - availableOversizedBuffers
- String buffer pool statistics under cn=StringBufferPool:
  - numTries
  - numHits
  - numMisses
  - numReleases
  - availableBuffers
- Vector pool statistics under cn=VectorPool:
  - numTries
  - numHits
  - numMisses
  - numReleases
  - availableBuffers

Work Queue

Incoming requests from clients are stored by connection handler threads in the work queue. The requests are then processed by the worker thread. The cn=Resource entry for the work queue has the following monitoring DN:

```
cn=Work Queue, cn=Resource, cn=instanceId, cn=Instance, cn=DPS60,
  cn=Product, cn=monitor
```

Entries below the cn=Resource entry for the work queue contain the following groups of attributes:

Settings

- maxNormalPriorityPeak identifies the maximum number of requests of normal priority that can be put in the queue. When this threshold is reached, the connection handler is suspended.
- maxHighPriorityPeak. identifies the maximum number of requests of high priority that can be put in the queue. When this threshold is reached, the connection handler is suspended.

State

- curNormalPriorityInQueue provides the current normal priority requests in queue.
- curHighPriorityInQueue provides the current high priority requests in queue.
- operationalStatus identifies the status of the element, with the following values: 2, and 4. For information about these values, see Table 24–2.

- statusDescription provides the detailed status description.

Statistics

- numNormalPriorityPuts identifies the number of requests of normal priority that are put in the queue by the connection handler threads.
- numNormalPriorityGets identifies the number of request of normal priority retrieved from the queue by worker threads.
- numHighPriorityPuts identifies the number of requests of high priority that are put in the queue by the connection handler threads.
- numHighPriorityGets identifies the number of request of high priority retrieved from the queue by worker threads.
- numAbandonRequests identifies the number of requests that are abandoned.
- numAbandonSuccesses identifies the number of requests that are abandoned while in the queue.

Worker Thread

The worker thread processes requests from the work queue.

The cn=Resource entry for the worker thread has the following monitoring DN:

```
cn=worker_thread_name, cn=Worker Thread, cn=Resource,
  cn=instanceId, cn=Instance, cn=DPS60, cn=Product, cn=monitor
```

Entries below the cn=Resource entry for the search thread contain the same groups of attributes as described in "Connection Handler Thread" on page 370, and the following attributes:

Statistics

- operationsProcessed identifies the number of operations processed by the worker thread.
- exceptionsCaught identifies the number of exceptions raised during the processing of operations.

Search Thread

When a search is performed on several data views, parallel search threads can be used. The cn=Resource entry for the search thread has the following monitoring DN:

```
cn=search_thread_name, cn=Search Thread, cn=Resource, cn=instanceId,
  cn=Instance, cn=DPS60, cn=Product, cn=monitor
```

Entries below the cn=Resource entry for the search thread contain the same groups of attributes as described in "Connection Handler Thread" on page 370.

Monitor Thread

The monitor thread checks the availability of remote data sources. A remote data source is considered to be available when the monitor thread can create one connection to the remote data source. The cn=Resource entry for the monitor thread has the following monitoring DN:

```
cn=monitor_thread_name, cn=Monitor Thread, cn=Resource, cn=instanceId,
  cn=Instance, cn=DPS60, cn=Product, cn=monitor
```

Entries below the cn=Resource entry for the search thread contain the same groups of attributes as described in "Connection Handler Thread" on page 370, and the following groups of attributes:

Settings

- backendServer identifies the name of the monitored remote data source.
- checkInterval identifies the interval of time (in seconds) between two checks.
- additionalCheckType identifies additional checking. The following values can be used:
  - 1 (no additional checks)
  - 2 (create a bind connection to the data source)
  - 3 (create a read connection to the data source)

State

- serverAvailable identifies the status of the remote data source. The value is true if the remote data source is up and running.

Statistics

- totalChecks identifies the total number of checks.
- availabilityChecksFailed identifies the number of failed availability checks. An availability check is successful when a remote data source is up and running.
- additionalChecksFailed identifies the number of failed additional checks.

Detailed Layout of the cn=monitor Entry

This section provides a detailed layout of the cn=monitor subtree.

```
cn=monitor
|
+-- cn=Product (Installed Product)
    |
    +-- cn=ProductName
```

```
|| setting:
|| - version
|| - buildNumber
|| - patchId
+-- cn=Operating System
    || setting:
    || - operatingSystemName
    || - operatingSystemVersion
    || state:
    || - (empty)
    || statisitics:
    || - (empty)
+-- cn=Instance  (Application System)
    |
    +-- cn=InstanceId (= host:port:instanceDir)
        |
        +-- cn=Service
            +-- cn=Add
                || statistics:
                || - total
                || - succeeded
                || - failed
                || - abandonned (?)
            +-- cn=Search
                || (same as Add operation above)
            +-- cn=Delete
            +-- cn=Compare
            +-- cn=Modify
            +-- cn=ModifyDN
            +-- cn=Bind
        +-- cn=SAP  (Service Access Point)
            +-- cn=listenerThread
                || settings:
                || - name
                || - isSecure
                || - host (?)
                || - port (?)
                || state:
                || - enabled
                || - operationalStatus
                || - statusDescription
                || - startTime
                || - stopTime
                || - stopException
                || statistics:
                || - acceptedConnections
                || - refusedConnections
    +-- cn=listenerThread
        || (same as above)
+-- cn=RSAP  (Remote SAP)
    +-- cn=LDAP Server servername
        || settings:
        || - name
        || - isSecure
        || - host (?)
        || - port (?)
        || state:
        || - operationalStatus
        || - statusDescription
        || - started
        || - readOnly
        || statistics:
        || - totalConnections
        || - totalAvailableConnections
        || - totalBindConnections
        || - availableBindConnections
        || - bindConnectionsRequested
        || - bindConnectionsProvided
        || - bindConnectionsRefused
        || - bindConnectionsWaitsRequired
        || - bindConnectionsReturnedValid
        || - bindConnectionsReturnedInvalid
        || - (idem for readConnections)
        || - (idem for writeConnections)
    +-- cn=LDAPS Server servername
        || (same as LDAP Server above)
    +-- cn=RDBM Server servername
        || settings:
        || - TBC
        || state:
        || - TBC
        || statistics:
        || - TBC
+-- cn=Component  (Logical Component)
    +-- cn=DataSource Pool poolname
        +-- cn=Proportional LB
            || settings:
            || - classname
            || state:
            || - enabled
            || statistics:
            || - totalBindConnectionsProvided
            || - totalBindConnectionsRefused
            || - totalAddConnectionsProvided
            || - totalAddConnectionsRefused
            || - totalCompareConnectionsProvided
            || - totalCompareConnectionsRefused
            || - totalDeleteConnectionsProvided
            || - totalDeleteConnectionsRefused
            || - totalModifyConnectionsProvided
            || - totalModifyConnectionsRefused
            || - totalModifyDNConnectionsProvided
            || - totalModifyDNConnectionsRefused
            || - totalCompareConnectionsProvided
            || - totalCompareConnectionsRefused
            +-- cn=Add
                || settings:
                || - (empty)
                || status:
                || - operationalStatus
                || - statusDescription
                || statistics:
                || - providedConnections
                || - providedPercentage
                || - refusedConnections
                || - refusedPercentage
            +-- cn=Search
                || (same as Add operation above)
            +-- cn=Delete
            +-- cn=Compare
            +-- cn=Modify
            +-- cn=ModifyDN
            +-- cn=Bind
+-- cn=JVM
    || settings:
    || - version
    || - jvmInstallation
    || state:
    || - operationalStatus
    || - statusDescription
    || statistics:
    || - totalJVMMemory
    || - maxJVMMemory
    || - freeJVMMemory
    || - realFreeJVMMemory
    || - JVMMemoryLowLevelCount
    || - JVMMemoryVeryLowLevelCount
    || - availableCPU
+-- cn=Resource  (System Resource)
    +-- cn=Worker Thread
        +-- cn=worker_thread_name
        || settings:
        || - threadID
        || - threadStack
        || state:
        || - operationalStatus
        || - statusDescription
        || - startTime
        || - started
        || - running
        || statistics:
        || - operationsProcessed
        || - exceptionsCaught
        +-- cn=Byte Buffer Pool
            || statistics:
            || - numTries
            || - numHits
            || - numMissesEmptyPool
            || - numMissesBufferSize
            || - numReleases
            || - availableStandardBuffers
            || - availableOversizedBuffers
        +-- cn=String Buffer Pool
            || statistics:
            || - numTries
            || - numHits
            || - numMisses
            || - numReleases
            || - availableBuffers
        +-- cn=Vector Pool
            || statistics:
            || - numTries
            || - numHits
            || - numMisses
            || - numReleases
            || - availableVectors
+-- cn=Search Thread
    +-- cn=search_thread_name
        || settings:
        ||
        || state:
        || - operationalStatus
        || - startTime
        || - stopTime
        || statistics:
        ||
        +-- cn=Byte Buffer Pool
            || (see Worker Thread)
```

```
                            +-- cn=String Buffer Pool
                                || (see Worker Thread)
                            +-- cn=vector Pool
                                || (see Worker Thread)
                    +-- cn=Monitor Thread
                        +-- cn=monitor_thread_name
                            || settings:
                            || - started
                            || - running
                            || - startTime
                            || - threadID
                            || - threadStack
                            || - backendServer
                            || - checkInterval
                            || - additionalCheckType
                            || state:
                            || - operationalStatus
                            || - statusDescription
                            || - serverAvailable
                            || statistics:
                            || - totalChecks
                            || - availabilityChecksFailed
                            || - additionalChecksFailed
                            +-- cn=Byte Buffer Pool
                                || (see Worker Thread)
                            +-- cn=String Buffer Pool
                                || (see Worker Thread)
                            +-- cn=vector Pool
                                || (see Worker Thread)
                    +-- cn=Connection Handler Thread
                        +-- cn=connection_handler_thread_name
                            || settings:
                            || - threadID
                            || - threadStack
                            || state:
                            || - operationalStatus
                            || - startTime
                            || - started
                            || - running
                            || - statusDescription
                            || statistics:
                            || - (empty)
                            +-- cn=Byte Buffer Pool
                                || (see Worker Thread)
                            +-- cn=String Buffer Pool
                                || (see Worker Thread)
                            +-- cn=Vector Pool
```

```
                || (see Worker Thread)
    +-- cn=Work Queue
        || settings:
        || - maxNormalPriorityPeak
        || - maxHighPriorityPeak
        || - operationalStatus
        || - statusDescription
        || state:
        || - curNormalPriorityInQueue
        || - curHighPriorityInQueue
        || statistics:
        || - numNormalPriorityPuts
        || - numNormalPriorityGets
        || - numHighPriorityPuts
        || - numHighPriorityGets
        || - numAbandonRequests
        || - numAbandonSuccesses
+-- cn=DB System
+-- cn=DB Service
```

◆ ◆ ◆ CHAPTER 25

Directory Proxy Server File Reference

This chapter describes the files found after you install Directory Proxy Server, and after you create server instances.

The examples shown in this chapter are for Solaris systems. File extensions and path separators may differ for your operating system. This chapter includes the following sections.

- "Software Layout for Directory Proxy Server" on page 381
- "Directory Proxy Server Instance Default Layout" on page 384

If you installed software from native packages, you may also use the packaging commands on your system to list the files installed. For example, after installing from native packages on Solaris systems, you can obtain a full list for a particular package using the `pkgchk -v` *package-name* command.

If you installed software from a zip distribution, find lists of installed files in the *install-path*`/dsee6/data/` directory.

Software Layout for Directory Proxy Server

This section describes the file layout you find after installing Directory Proxy Server from the zip distribution. All files locations are relative to the path where you installed the product. For information on default native package installation locations, see "Default Paths and Command Locations" on page 24.

install-path`/dps6/`

Directory Proxy Server files shared by server instances. This directory houses the following files of interest.

install-path`/dps6/bin/dpadm`

Directory Proxy Server command for local administration. See `dpadm`(1M).

install-path`/dps6/bin/dpconf`

Directory Proxy Server command for configuration over LDAP. See `dpconf`(1M).

install-path`/dps6/etc/`
Directory Proxy Server configuration templates, not intended to be used directly.

install-path`/dps6/examples/`
Sample Directory Proxy Server plug-ins (currently empty).

install-path`/dps6/lib/`
Shared Directory Proxy Server libraries, not intended for use directly.

install-path`/dps6/resources/`
Directory Proxy Server resource files, not intended to be used directly.

install-path`/dscc6/`
Directory Service Control Center agent files shared by multiple Directory Server Enterprise Edition component products. This directory houses the following files of interest.

install-path`/dscc6/bin/dsccmon`
Command to monitor servers managed through Directory Service Control Center. See `dsccmon`(1M).

install-path`/dscc6/bin/dsccreg`
Command to manage the Directory Service Control Center registry. See `dsccreg`(1M).

install-path`/dscc6/bin/dsccsetup`
Command to set up Directory Service Control Center. See `dsccsetup`(1M).

install-path`/dscc6/etc/`
Directory Service Control Center agent configuration information, not intended to be used directly.

install-path`/dscc6/lib/`
Shared libraries, not intended to be used directly.

install-path`/dsee6/`
Files shared by multiple Directory Server Enterprise Edition component products. This directory houses the following files of interest.

install-path`/dsee6/bin/certutil`
NSS certificate manipulation command used by other tools, not intended to be used directly.

install-path`/dsee6/bin/dsee_deploy`
Command to install and remove software. See `dsee_deploy`(1M).

install-path`/dsee6/bin/ldif`
Command to base64 encode LDIF attribute values. See `ldif`(1)

install-path`/dsee6/cacao_2.0/`
Common agent container files shared by Directory Server Enterprise Edition component products, not intended to be used directly.

install-path`/dsee6/data/`
Lists of installed files used by the `dsee_deploy` command, not intended to be used directly.

install-path`/dsee6/lib/`
Libraries shared by Directory Server Enterprise Edition component products, not intended to be used directly.

install-path`/dsee6/man/`
Directory Server Enterprise Edition online reference manual pages. See also *Sun Java System Directory Server Enterprise Edition 6.3 Man Page Reference*.

install-path`/dsee6/private/include/`
Directory SDK for C header files used by Directory Server Enterprise Edition component products.

install-path`/dsee6/private/lib/`
Directory SDK for C shared libraries used by Directory Server Enterprise Edition component products.

install-path`/dsrk6/bin/ldapcmp`
Directory Server Resource Kit command to compare LDAP entries from two directories. See `ldapcmp`(1).

install-path`/dsrk6/bin/ldapcompare`
Directory Server Resource Kit command to perform LDAP compare operations. See `ldapcompare`(1).

install-path`/dsrk6/bin/ldapdelete`
Directory Server Resource Kit command to delete directory entries. See `ldapdelete`(1).

install-path`/dsrk6/bin/ldapmodify`
Directory Server Resource Kit command to update entries over LDAP. See `ldapmodify`(1).

install-path`/dsrk6/bin/ldappasswd`
Directory Server Resource Kit command to change user passwords. See `ldappasswd`(1).

install-path`/dsrk6/bin/ldapsearch`
Directory Server Resource Kit command to search a directory. See `ldapsearch`(1).

install-path`/dsrk6/lib/`
Libraries used by Directory Server Resource Kit commands, not intended to be used directly.

install-path`/jre/`
Java Runtime Environment, not intended to be used directly.

install-path`/var/`
Container for runtime files, not intended to be used directly.

Directory Proxy Server Instance Default Layout

This section describes the file layout you find after creating a Directory Proxy Server instance. The *instance-path* is the file system path where you created the instance.

instance-path`/alias/`
Certificate database files, not intended to be used directly.

instance-path`/config/`
Server configuration files, not intended to be used directly.

instance-path`/etc/`
Additional instance configuration, not intended to be used directly.

instance-path`/logs/`
Default server logs directory. The following files are stored here.

`access` logs	This file records information about the requests processed by Directory Proxy Server. For detail about `access` logs, see "Access Logs for Directory Proxy Server" on page 348.
`errors` logs	This file records errors, warnings, and informational messages logged during Directory Proxy Server operation. For detail about `errors` logs, see "Error Logs for Directory Proxy Server" on page 345.

instance-path`/tmp/`
Server runtime files directory, not intended to be used directly.

APPENDIX A

Directory Server Resource Kit File Reference

This appendix describes the files found after you install Directory Server Resource Kit. You can also find lists of installed files in the *install-path*`/dsee6/data/` directory.

The examples shown in this appendix are for Solaris systems. File extensions and path separators may differ for your operating system.

Software Layout for Directory Server Resource Kit

This section describes the file layout you find after installing Directory Server Resource Kit from the zip distribution. All files locations are relative to the path where you installed the product. For information on default native package installation locations, see "Default Paths and Command Locations" on page 24.

install-path`/dsee6/`

Files shared by multiple Directory Server Enterprise Edition component products. This directory houses the following files of interest.

install-path`/dsee6/bin/certutil`

NSS certificate manipulation command used by other tools, not intended to be used directly.

install-path`/dsee6/bin/dsee_deploy`

Command to install and remove software. See `dsee_deploy`(1M).

install-path`/dsee6/bin/ldif`

Command to base64 encode LDIF attribute values. See `ldif`(1)

install-path`/dsee6/cacao_2.0/`

Common agent container files shared by Directory Server Enterprise Edition component products, not intended to be used directly.

install-path`/dsee6/data/`
Lists of installed files used by the `dsee_deploy` command, not intended to be used directly.

install-path`/dsee6/lib/`
Libraries shared by Directory Server Enterprise Edition component products, not intended to be used directly.

install-path`/dsee6/man/`
Directory Server Enterprise Edition online reference manual pages. See also *Sun Java System Directory Server Enterprise Edition 6.3 Man Page Reference*.

install-path`/dsrk6/bin/authrate`
Directory Server Resource Kit command to measure authentication rate. See `authrate`(1).

install-path`/dsrk6/bin/dsmlmodify`
Directory Server Resource Kit command to update entries using DSML v2. See `dsmlmodify`(1).

install-path`/dsrk6/bin/dsmlsearch`
Directory Server Resource Kit command to search a directory using DSML v2. See `dsmlsearch`(1).

install-path`/dsrk6/bin/example_files/`
Sample template files for use with the `makeldif`.

install-path`/dsrk6/bin/ldapcmp`
Directory Server Resource Kit command to compare LDAP entries from two directories. See `ldapcmp`(1).

install-path`/dsrk6/bin/ldapcompare`
Directory Server Resource Kit command to perform LDAP compare operations. See `ldapcompare`(1).

install-path`/dsrk6/bin/ldapdelete`
Directory Server Resource Kit command to delete directory entries. See `ldapdelete`(1).

install-path`/dsrk6/bin/ldapmodify`
Directory Server Resource Kit command to update entries over LDAP. See `ldapmodify`(1).

install-path`/dsrk6/bin/ldappasswd`
Directory Server Resource Kit command to change user passwords. See `ldappasswd`(1).

install-path`/dsrk6/bin/ldapsearch`
Directory Server Resource Kit command to search a directory. See `ldapsearch`(1).

install-path`/dsrk6/bin/ldapsubtdel`
Directory Server Resource Kit command to delete a directory subtree recursively. See `ldapsubtdel`(1).

install-path`/dsrk6/bin/ldifxform`
Directory Server Resource Kit command to transform LDIF content. See `ldifxform`(1).

install-path`/dsrk6/bin/logconv`
Directory Server Resource Kit command to analyze Directory Server access logs. See `logconv`(1).

install-path`/dsrk6/bin/makeldif`
Directory Server Resource Kit command to generate LDIF content for testing and benchmarking purposes. See `makeldif`(1).

install-path`/dsrk6/bin/modrate`
Directory Server Resource Kit command to measure rates of modifications to directory entries. See `modrate`(1).

install-path`/dsrk6/bin/searchrate`
Directory Server Resource Kit command to measure search rates. See `searchrate`(1).

install-path`/dsrk6/class/`
Libraries used by Directory Server Resource Kit commands, not intended to be used directly.

install-path`/dsrk6/lib/`
Libraries used by Directory Server Resource Kit commands, not intended to be used directly.

Index